Universitat
Autònoma
de Barcelona

4219: Acceleració d'operacions SQL en GPU

Memòria del Projecte Fi de Carrera
d'Enginyeria en Informàtica
realitzat per
Jaume Llunell Gómez
i dirigit per
Juan Carlos Moure López
Bellaterra, 12 de Setembre de 2011

Universitat Autònoma de Barcelona

Escola Tècnica Superior d'Enginyeria

El sotasignat, Juan Carlos Moure López

Professor/a de l'Escola Tècnica Superior d'Enginyeria de la UAB,

CERTIFICA:

Que el treball a què correspon aquesta memòria ha estat realitzat sota la seva direcció per en

Jaume Llunell Gómez

I per tal que consti firma la present.

Signat: Juan Carlos Moure López

Bellaterra, 12 de Setembre de 2011

Índex de continguts

1 Introducció

En l'actualitat, les dades es guarden en bases de dades per estructurar-les i permetre que siguin accessibles i modificables amb coherència. D'altre banda, cada vegada s'ha de treballar amb un volum de dades major a causa de diversos factors: el creixement dels usuaris en les noves aplicacions que generen informació de forma continua o l'emmagatzematge de dades sense utilitat operativa que ara son destinades a estudis de mercat o a l'estadística, entre altres.

A causa d'aquest creixement i de les noves demandes en les aplicacions, els tradicionals sistemes de gestió de bases de dades (SGBD) com Oracle o SQL Server han acabat sent desbancats per altres de nova creació que s'adapten millor a problemes concrets (gestió de documents, utilització de dades distribuïdes , bases de dades multimèdia, etc). Com es pot veure en el següent esquema (figura 1.1), els antics SGBD (requadre verd) han anant perdent protagonisme en certs àmbits especialitzats a favor d'altres que tenen millor rendiment en aquestes situacions.

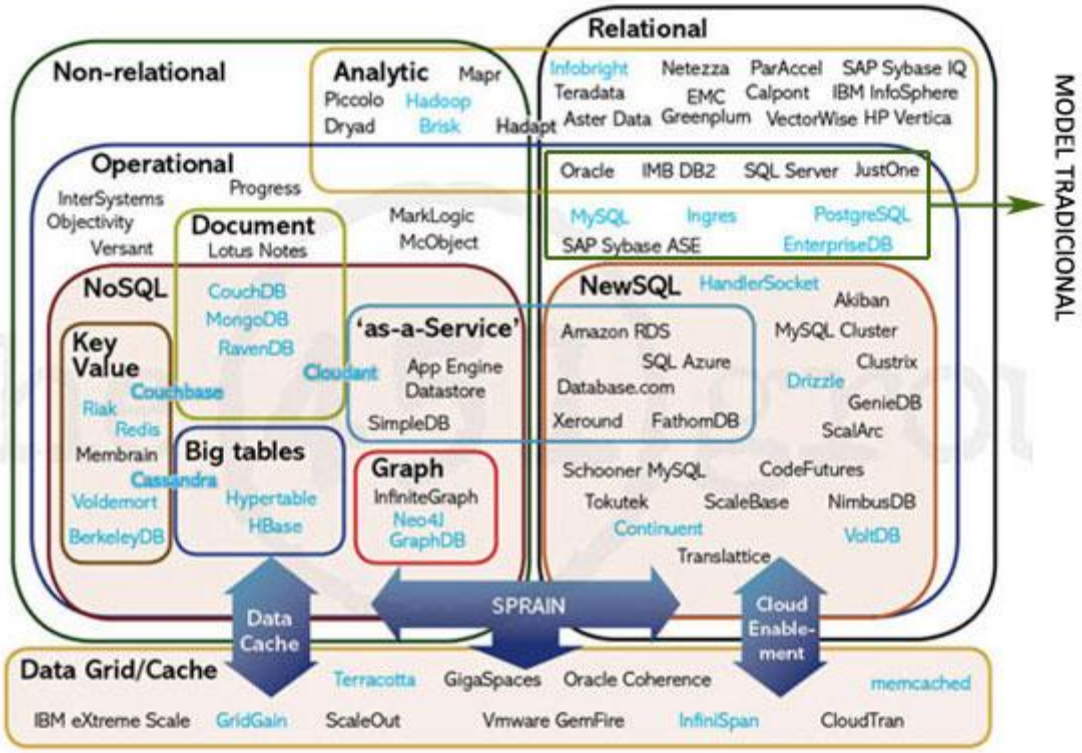


Figura 1.1: Panoràmica de la complexa situació actual. En concret, al requadre verd hi ha els tradicionals SGBD. La resta son sistemes especialitzats en un problema (en groc els analítics).

El problema concret que es tractarà en aquest projecte és el de les consultes analítiques . Aquestes solen ser construïdes a demanda per satisfer, al moment, la petició d'informació especifica. Es solen anomenar consultes *ad-hoc* ("per això") perquè només

son útils per a un determinat propòsit i no poden ser considerades en el disseny inicial de la base de dades. Tot això fa que no es pugui optimitzar la base de dades a priori per obtenir un temps d'execució ràpid amb aquest tipus de consultes. A més a més, es veu agreujat en el moment en que la mida de la base de dades creix i la informació entre la que buscar és molt més extensa. Per tant ens trobarem en situacions en que el temps de resposta s'incrementa molt.

Per donar solució a aquests problemes, s'avaluarà la possibilitat d'utilitzar la tecnologia GPU de Nvidia. La gran paral·lelització obtinguda sobre un esquema vectorial d'atributs de bases de dades és el punt de partida per començar a reduir els temps de resposta dels sistemes actuals amb aquest tipus de hardware.

Tot i així, al resoldre problemes específics ens trobem amb inconvenients que no s'aconsegueixen solucionar. En el nostre cas, tot i que no es consideraran, les escriptures i modificacions es comportarien pitjor comparades amb bases de dades tradicionals.

1.1 Motivacions

El camp de bases da dades sempre ha estat present en tots els treballs pràctics on he participat, ja que tota aplicació desenvolupada en tenia una com a capa d'emmagatzematge. Tot i així, sempre les havia tractat d'una forma superficial sense poder aprofundir en els aspectes de rendiment.

La falta de coneixements sobre les implementacions reals dels SGBD fa que al produir-se un problema o un temps de resposta no esperat, no es puguin trobar solucions reals per culpa de la falta de diagnòstic.

Degut això, vaig intentar enfocar el meu projecte cap a aquest sector, ja que em permetria poder estudiar les principals característiques de les bases de dades actuals i continuar aprofundint a través d'un hardware versàtil i en creixement com son les GPU. D'aquesta manera podria obtenir una visió general d'aquest àmbit.

1.2 Objectius

Els objectius marcats pel projecte son:

- Estudi de les implementacions bàsiques realitzades en els motors de bases de dades reals en entorn de treball *multicore* per poder fer operacions relacionals de lectura de dades.
- Estudi de les estructures de memòria i execució de processadors *manycore* (GPUs) per avaluar algorismes que implementin les operacions tractades en entorns *muticore*.
- Execució d'un joc de probes estàndard sobre diferents SGBD per obtenir els seus temps de resposta en consultes i les seves estratègies d'execució.
- Avaluació dels resultats obtinguts per poder determinar les millores i pèrdues en guany de realitzar aquestes operacions en GPUs.

- Comprensió en profunditat els algorismes interns de les operacions relacionals per poder obtenir un millora en el disseny de bases de dades.
- Anàlisis i possibles propostes de millora dels diferents treballs d'investigació ja realitzats sobre el camp d'acceleració d'operacions SQL en GPUs.

1.3 Viabilitat del projecte

1.3.1 Especificacions del projecte

Per l'avaluació de les diferents alternatives s'haurà de fixar un entorn de proves. Per tant inicialment s'especificaran unes consultes i un disseny de base de dades únic, que s'utilitzarà per avaluar els diferents sistemes presentats. També es buscaran diferents càrregues de dades per poder obtenir diferents resultats i poder extreure'n les conclusions pertinents.

1.3.2 Viabilitat tècnica

Per les diferents alternatives s'utilitzarà C/C++ en entorns de programació ja disponibles. En els entorns *manycore* s'utilitzarà el SDK proporcionat per Nvidia.

En la majoria de casos les implementacions ja estan realitzades i provades pels autors, per tant només s'haurà d'avaluar i analitzar.

L'entorn de treball serà Microsoft Windows XP Professional x64.

1.3.3 Viabilitat operativa

El treball a realitzar pot ser fet únicament per l'estudiant, on a través de les tutories i consultes externes es podran resoldre els dubtes i validar els resultats que s'aconsegueixin.

1.3.4 Viabilitat econòmica

Les inversions inicials previstes son la compra d'un dispositiu de Nvidia de *manycores*. El seu preu aproximat es de 120€. Per altre banda, totes les eines utilitzades son ja disponibles gratuïtament ja sigui per entorns comercials o educatius.

1.4 Planificació del projecte

Es resumeixen en els punts següents les tasques previstes per poder complir els objectius marcats :

- Estudi de les característiques i aplicacions de les GPU i CUDA.
- Estudi de les possibles implementacions fetes en bases de dades i GPU.
- Recerca de les implementacions i alternatives en SGBD actuals.
- Estudi de les necessitats actuals de les aplicacions que usen SGBD.
- Elecció i fixació dels entorns de probes i les eines de profiling.
- Comprensió del codi font de l'aplicació escollida i avaluació de les decisions fetes pels creadors.

- Realització de tasques d'avaluació de l'aplicació.
- Anàlisi dels els resultats obtinguts.
- Recerca i avaluació de millores si es produeixen resultats no satisfactoris.
- Anàlisis final un cop acabades les modificacions.
- Documentació de les feines realitzades i presentació.

A continuació es descriu la planificació del projecte, condicionada al resultats obtinguts dels experiments i de la recerca realitzada.

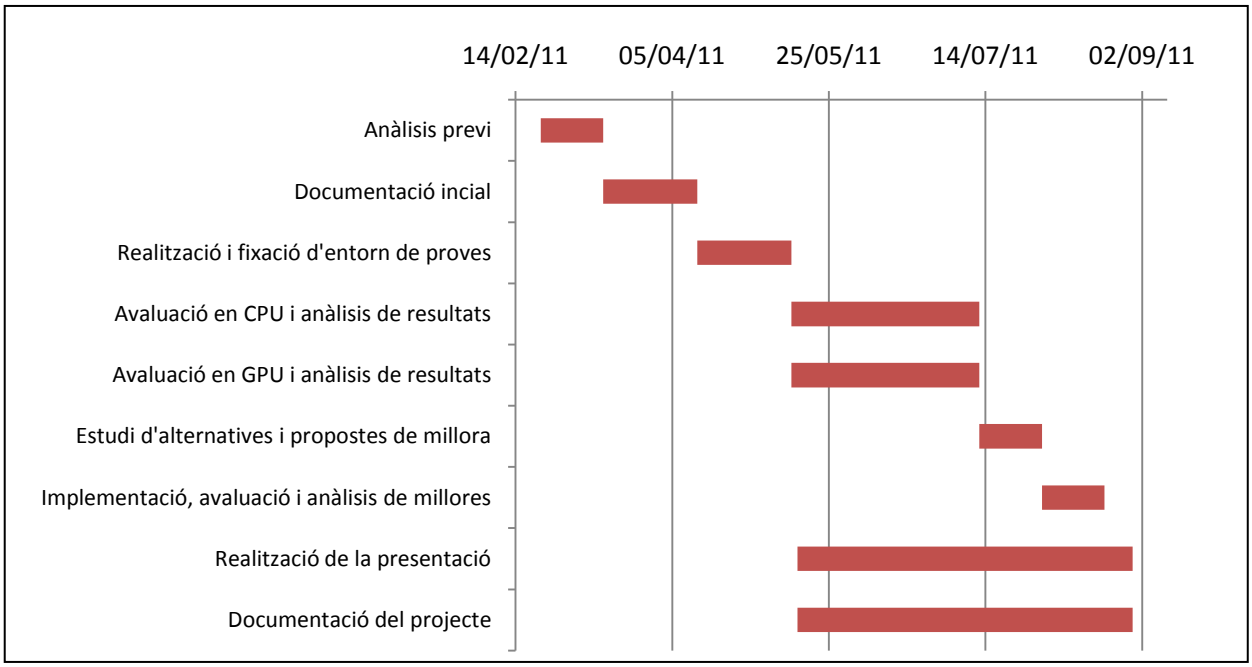


Figura 1.2: Planificació inicial del projecte desenvolupat.

Tot seguit es descriu l'esquema seguit en la memòria:

1. **Introducció:** Es contextualitzen les bases de dades actuals. Es planteja la viabilitat del projecte i es planifiquen les tasques a realitzar.

2. **Marc teòric:** S'exposen els conceptes teòrics inicials des d'on parteix el projecte, així com les diferents alternatives escollides per la seva realització.

3. **Metodologies de profiling i eines d'avaluació:** S'especifiquen les eines hardware i software utilitzades per obtenir els resultats i hipòtesis posteriors.

4. **Avaluació dels sistemes de bases de dades:** S'exposen els resultats obtinguts amb les eines de profiling de totes les alternatives escollides.

5. **Estudi dels resultats:** S'aprofundeix en els resultats anteriors per poder explicar el seu comportament.

6. **Conclusions :** S'exposen les conclusions a les que hem arribat després de les avaluacions i estudis realitzats.

2 Marc teòric

El marc teòric introduirà els conceptes bàsics de l'estructura interna d'una base de dades que permet executar consultes. Repassarà les possibilitats actuals dels SGBD en CPU i es posarà en el context de les GPU i el seu ús en bases de dades.

2.1 Àlgebra relacional

Per poder operar sobre les relacions necessitem un llenguatge relacional. Aquest llenguatge serà interpretat i produirà un seguit d'operacions fonamentals sobre una o varies relacions, com ara la unió, la diferencia o el producte cartesià ... el resultat dels quals també serà una relació.

Unió ($R_1 \cup R_2$)

Realitza una unió de totes les tuples de R_1 i R_2, i el resultat serà una altre relació on cada tupla estarà en R_1 , en R_2 o en les dos.

Diferencia ($R_1 - R_2$)

Realitza una resta de totes les tuples de R_2 sobre R_1, i el resultat serà una altre relació on s'eliminaran de R_1 totes les tuples que estiguin a R_2.

Producte cartasià ($R_1 \times R_2$)

Realitza una concatenació dels atributs de R_1 i R_2, i el resultat seran totes les possibles tuples que es poden generar a partir de concatenar cada tupla de R_1 amb totes les tuples de R_2.

Selecció ($\sigma_p(R_1)$)

Realitza un selecció de totes les tuples de R_1 que compleixen una certa condició booleana p.

Projecció ($\Pi_s(R_1)$)

Realitza una reducció de la relació R_1 a un cert subconjunt d'atributs de R_1. S'eliminen els duplicats.

Join ($R_1 \bowtie R_2$)

Operació que podria expressar a través d'operacions anteriors, però expressada així particularment per la seva utilitat pràctica. Realitza un producte cartesià sobre les

relacions R_1 i R_2, però nomes mante aquelles que satisfan la igualtat en els atributs comuns de R_1 i R_2. Es pot veure un exemple a la figura 2.1.

alumnes

alumneID	*nom*	*Edat*
1	Albert	23
2	Maria	22
3	Joan	21

reserves

reservaId	*alumneId*	*aula*	*hora*
1	1	aula1	22/04/11 5pm
2	2	aula2	23/04/11 3pm
3	2	aula1	25/04/11 4pm

alumnes ⋈ reserves

alumneID	*nom*	*edat*	*reservaId*	*aula*	*hora*
1	Albert	23	1	aula1	22/04/11 5pm
2	Maria	22	2	aula2	23/04/11 3pm
2	Maria	22	3	aula1	25/04/11 4pm

Figura 2.1: Operació join per les relacions alumnes i reserves

2.3 Processament de consultes

2.3.1 SQL

La gran majoria dels sistemes de gestió de bases de dades utilitzen SQL (Structured Query Language) com a llenguatge per consultar i modificar una base de dades. Aquest llenguatge té capacitats molt properes a l'àlgebra relacional i a més, afegeix altres operacions de consulta que han demostrat ser molt útils en aplicacions pràctiques, com ara la ordenació, les operacions d'agregació o l'eliminació de duplicats.

2.3.2 Interpretació de consultes i generació del pla d'execució

Les consultes SQL s'han d'interpretar per poder produir els resultats esperats. Així doncs, el complex model disposa de varis elements que realitzen diferents tasques per interpretar el llenguatge SQL i produir un pla d'execució.

Disposen d'un *parser* per poder traduir les consultes a un llenguatge intern. Aquest llenguatge és enviat a un planejador d'execució, que determinarà quines operacions s'han de realitzar amb cada taula i en quin ordre per poder retornar el resultat esperat.

Per últim, abans d'executar les operacions, es passa el pla d'execució obtingut a l'optimitzador, que s'encarregarà de variar el pla inicial per intentar aconseguir una execució òptima, mitjançant el catàleg del sistema (índex disponibles, mida de les taules, etc...). Un cop tenim el pla, s'executa l'avaluador d'operacions.

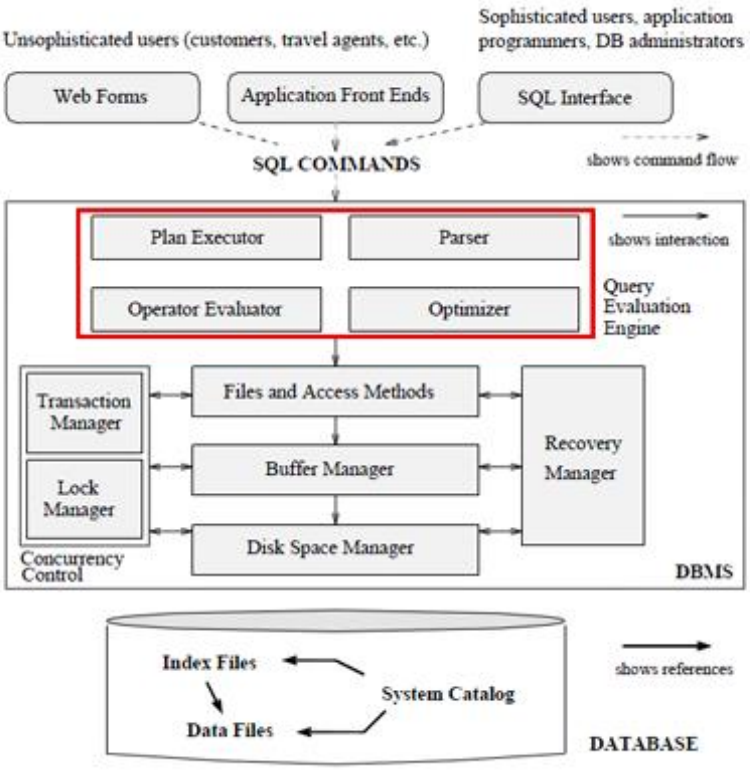


Figura 2.2: Arquitectura d'un sistema de gestió de base de dades.

2.3.3 Execució de consultes

El pla d'execució de consultes en base de dades pot ser representat en forma d'arbre, com es mostra en la figura 2.3. D'aquesta manera podrem executar la consulta i retornar el resultat executant les operacions de les fulles inferiors de l'arbre i ascendir fins el node principal, obtenint les tuples resultants.

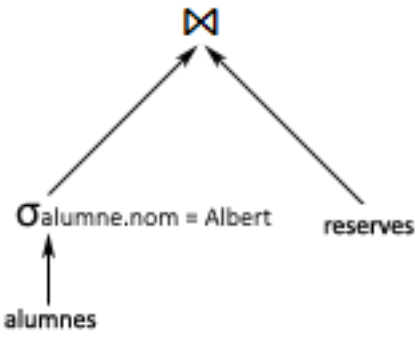


Figura 2.3: Operació join per les relacions alumnes i reserves

El problema d'aquesta forma d'avaluar consultes, és que cada operació ha de materialitzar els seus resultats entremitjos a la memòria RAM. Hi ha vegades que no és necessària aquesta materialització i es pot executar en forma de *pipeline* en que una operació consumeix directament els resultats d'una altre operació productora. D'aquesta manera podrem reduir el nombre de resultats que han de ser guardats a la memòria i reduir el nombre d'operacions E/S.

2.3.4 Catàleg del sistema

Totes les operacions del punt anterior s'executen amb diferents algorismes implementats en els sistemes de gestió de bases de dades. A més a més, es posen a disposició dels administradors i dissenyadors algunes funcionalitats especials per les taules. Com a resultat es poden indexar els atributs d'una taula a través d'un índex (*B+Tree* o *Hash*) per accelerar les consultes d'uns determinats camps que son usats amb freqüència. D'aquesta forma els patrons d'accés al disc que obtenim a través d'aquestes optimitzacions, ens permetran executar les operacions primitives de diferents maneres: *Iterativament* si no es disposa de cap indexació, *indexant* si es disposa d'algun índex o *particionant* per subdividir el problema en petits problemes menys costosos.

2.4 Capa d'emmagatzematge

El model relacional és un model lògic i per tant no implica una implementació física determinada. Tot i així, encara que es presenten diferents alternatives per modelar físicament el model lògic [1], la major part dels sistemes de bases de dades adapten la mateixa estructura per emmagatzemar la informació.

clientID	*dni*	*nom*	*edat*
1	40352684N	John	29
2	77618880K	Mary	28
3	46212096N	Tom	32
4	33180832A	Albert	35

Figura 2.4 : Exemple d'un model relacional d'una base de dades consistent en una taula, clients.

2.4.1 NSM (N-ary storage model)

En el model d'emmagatzematge n-ary (on n es el nombre d'atributs en la relació) es disposa d'un fitxer per cada taula on s'hi guarden les tuples, o registres, un darrera l'altre, habitualment distribuïts en diferents blocs de disc de mida fixa.

El *header* de pàgina s'utilitza per emmagatzemar dades comunes a tota la pàgina, com ara apuntadors als espais lliures o apuntadors a la següent pàgina, entre altres.

Degut a que alguns dels atributs de les tuples poden tenir longitud variable, es pot fer servir un *header* de tupla on es codificarà la llargada juntament amb altra informació útil depenent de la implementació.

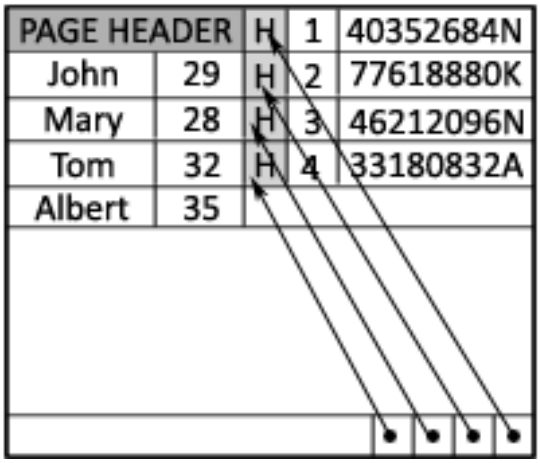


Figura 2.5 : Capa d'emmagatzematge NSM per la relació client

Per altra banda, al final de la pàgina es mantindrà una llista d'apuntadors a les tuples, on el n-essim element apunta a la n-essima tupla.

Aquesta estructura permet mantenir junt tot un registre, fet que beneficiarà les noves insercions o modificacions. També serà útil per mantenir, a baix cost , els mecanismes de bloqueig a nivell de bloc de disc o a nivell de tupla. Els bloquejos son necessaris per mantenir la base de dades consistent en cas de modificacions concurrents (com més específic siguin, més concurrència permetrem) ja que actuaran com a semàfors que habilitaran o negaran el permisos per modificar o insertar.

Els inconvenients d'aquest model d'emmagatzematge rauen principalment en situacions on no son necessaris tots els atributs de la taula. En NSM s'haurà de llegir del disc tots els atributs encara que no siguin necessaris. Conseqüentment, també es perdrà localitat espacial entre atributs i es produiran molts error de *cache* de CPU. Això esdevé en processos de lectura de dades, com poden ser les aplicacions de presa de decisions o consultes intenses. Aquestes aplicacions no solen necessitar bloquejos i per tant s'han d'adoptar nous esquemes d'emmagatzematge.

2.4.2 DSM (Decomposed storage model)

En DSM es descompon una relació en n atributs verticalment. Oposadament a NSM, els blocs de disc contenen només valors d'un mateix atribut. Igualment, en atributs amb mida variable, s'ha de seguir mantenint el desplaçament per aconseguir trobar els finals d'atributs a través d'apuntadors a final de pàgina.

DSM no pateix en els casos de només lectura ja que només ens cal llegir els blocs de disc de les columnes necessàries. També fa que els valors estiguin consecutivament, fet que disminueix els errors de *cache*. El problema d'aquesta distribució és en el moment de la recomposició total dels n-aris atributs, ja que s'haurà de reconstruir la tupla completament a partir de diferents pàgines de dades.

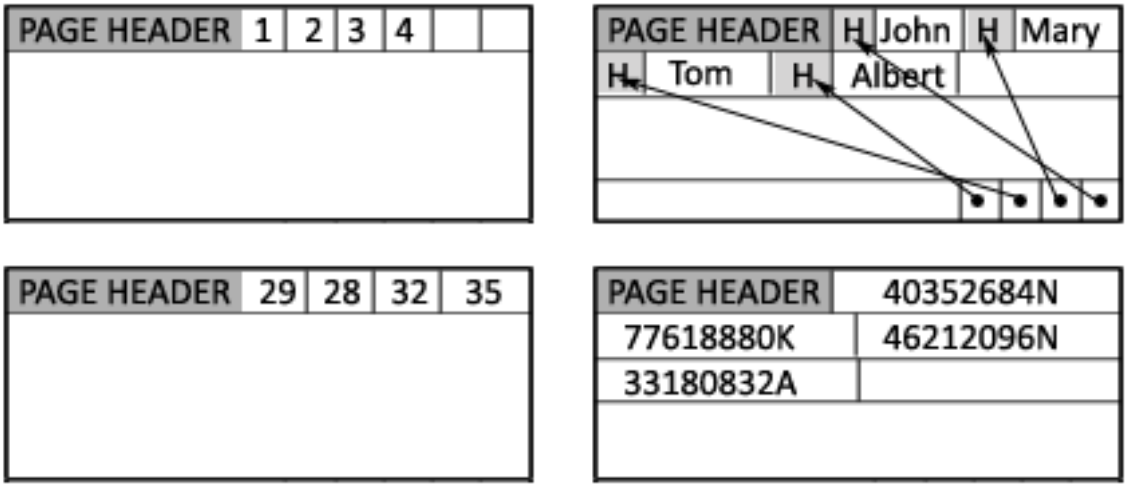


Figura 2.6 : Capa d'emmagatzematge DSM per la relació client

2.4.3 PAX (Partition attributes across)

La més recent estructura d' emmagatzematge és el PAX. En aquesta, les dades d'una mateix tupla s'emmagatzemen en una mateixa pàgina o bloc de disc, però es realitza una partició vertical organitzada en mini pàgines.

La combinació de NSM i DSM fa que en PAX poguem disposar de totes les dades d'una sola tupla en un mateix bloc de disc i que alhora no patim tants errors de *cache* i obtinguem localitat espacial. El problema recau en que, igual que en NSM, haurem de llegir tots els atributs de disc encara que només en necessitem pocs. D'aquesta forma tindrem el mateix *overhead* E/S.

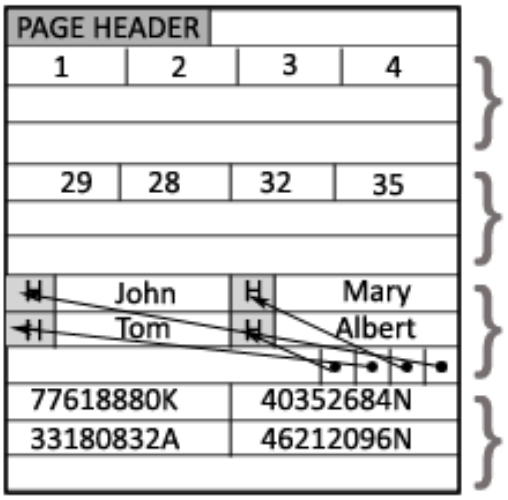


Figura 2.7 : Capa d'emmagatzematge PAX per la relació client

2.5 Entorns CPU (Single-Core i Multi-Core)

En aquest apartat es fa una breu introducció als diferents SGBD vistos per entorns CPU Single-Core i Multi-Core.

2.5.1 Sistemes orientats a la transacció.

Un SGBD ha de permetre a diferents usuaris realitzar operacions de lectura i escriptura concurrentment. A més a més, després de cada operació l'estat del sistema s'ha de mantenir consistent. Per aconseguir això s'han creat les transaccions. Aquestes representen un interacció atòmica amb la base de dades, es a dir, que es realitzen un seguit d'operacions com un bloc i en cas de fallar totes elles son cancel·lades deixant l'estat del sistema com al inici de la seva l'execució.

Les propietats bàsiques que han de complir les transaccions per aconseguir aquests objectius son les anomenades ACID (Atomicity, Consistency, Isolation, Durability).

Atomicitat: Requereix que les modificacions de base de dades segueixin la norma de "tot o res". És a dir que una transacció s'ha de realitzar completament o s'ha de cancel·lar tot el que ha realitzat, ja que no pot existir una transacció incompleta. Això també hauria d'incloure errors del sistema com falles d'energia o de hardware, per evitar que el sistema resti en un estat inconsistent.

Consistència: Requereix que tota modificació porti la base de dades des d'un estat consistent a un altre estat consistent. Es a dir, que satisfacin totes les regles d'integritat que han estat creades pel dissenyador.

Aïllament: Requereix que una transacció ha de ser executada com si cap altre s'estigués executant en aquell mateix moment. En altres paraules, no és possible que dues transaccions s'interfereixin i modifiquin les mateixes dades al mateix moment. Aquesta propietat marcarà el nivell de concurrència que es permet, i es possible relaxar-la per obtenir millors resultats.

Durabilitat: Requereix que els efectes d'una transacció finalitzada no es poden perdre. Es a dir, que un cop s'ha validat una transacció les seves accions es mantindran encara que es produeixin falles del sistema.

La implementació de transaccions que compleixin aquestes propietats portaran a que un sistema de base de dades sigui fiable. També cal dir que, al ser un recurs fonamental per interactuar amb la base de dades, l'eficiència amb que s'apliquen aquestes propietats marcarà molt el rendiment final obtingut pel sistema, ja que normalment s'espera que es puguin executar el més gran nombre d'elles al mateix moment.

Per altre banda cal dir que aquests sistemes estan orientats a donar un bon servei tant de lectura com d'escriptura. Per tant, tradicionalment, solen organitzar les dades amb un model d'emmagatzematge N-ary (NSM) i solen ser utilitzades en entorns de producció amb uns requisits estàndards, com per exemple paquets de comptabilitat o facturació.

2.5.2 PostgreSQL

PostgreSQL és un sistema de base de dades multiplataforma de codi lliure que inicia el seu desenvolupament aproximadament el 1986 a partir del codi d'*Ingres*. Actualment és un dels sistemes amb més usuaris arreu del món i és utilitzada en producció per grans empreses.

Aconsegueix treballar en varis nuclis (Multi-Core) a nivell de connexió, però el processament de consultes es fa en forma Single-Core.

Es tracta d'un sistema orientat a la transacció, per tant el seu objectiu és obtenir la major concurrència de transaccions possible complint les propietats ACID. Això ho aconsegueix a través de:

El control de concurrència es realitza a través de multiversions (***Multiversion concurrency control o MVCC***). És a dir, que en comptes de realitzar les operacions d'escriptura de base de dades reescrivint els antics valors pels nous, es realitzen noves versions de les dades mantenint les antigues però fent-les obsoletes. Això permet a les transaccions veure una determinada versió de la base de dades en el seu moment d'execució i realitzar el control de consistència a través d'aquestes versions. Cal dir que les versions es realitzen a nivell de fila i quan cap transacció utilitza les antigues versions de dades, un procés s'encarrega d'eliminar-les. Generalment porta a un millor rendiment que l'altre alternativa, els *bloquejos.* Al treballar amb versions, les operacions de lectura no es troben ni realitzen bloquejos que no els deixen continuar. El principal desavantatge és la feina extra de mantenir i controlar tota l'estructura al disc.

El model en que emmagatzema les dades és NSM, on les files es guarden juntes i no es permet que alguns dels seus atributs quedin emmagatzemats en pàgines de dades diferents. La mida de pagina és de 8 KB i no es pot modificar.

Pel que fa a la indexació d'atributs, permet quatre formes de realitzar-los: En primer lloc, a través d'arbres balancejats ***B-Tree*** (per defecte). En segon lloc, i especialment si es realitzen recerques només per igualació i no es permeten els *nulls*, s'aconsella utilitzar l'índex ***Hash.*** També disposa de dos últimes formes d'indexació més complexes ***GIN*** i ***GIST***.

GIN (Generalized Inverted Index) emmagatzema una llista de claus. Cadascuna d'aquestes conté una llista de files en la que apareix aquesta clau. És útil per fer recerques per atributs de tipus vector o per buscar un text complert.

GIST (Generalized Search Tree), d'altre banda, genera un arbre balancejat *B-Tree* però permet definir com s'han de tractar les claus. Això possibilita realitzar recerques que no compleixen el patró de igualtat o rang, com per exemple les distancies.

Els algorismes *JOIN* permeten "*nested loop join*" , "*merge join*" i "*hash join*". En el primer s'utilitza en cas de tenir disponible un índex o si l'optimitzador ho veu convenient. Es realitza iterant sobre les taules afectades per mitjà d'un índex o seguint

un ordre seqüencial. En el segon ("*merge join*"), s'ordenen els conjunts i després es realitza la *join* iterativament. D'aquesta forma, al estar ordenats es pot abandonar la iteració en el moment en que ja s'ha superat el valor desitjat. El tercer ("*hash join*"), s'utilitza només en igualtats i fa servir taules *hash* per localitzar si existeixen files que compleixen la condició de *join*.

En els algoritmes d'ordenació, s'utlitza "*quicksort*" si es pot realitzar en memòria o "*external merge sort"* si s'ha d'acabar realitzant a disc.

Pel que fa a la capacitat de memòria que permet utilitzar, no imposa topall màxim i es configurable mitjançant paràmetres. Permet fixar uns valors de memòria total que mai es podrà superar i després dividir aquest valor per diferents utilitats (*cache* de pagines i algorismes d'ordenació).

També permet, com la majoria de sistemes, claus primàries, claus foranies, claus úniques, *triggers, checks* i procediments emmagatzemats. [2]

2.5.3 Firebird

Firebird és un sistema de bases de dades multiplataforma que s'inicia amb l'alliberació del codi de Interbase (Borland) al 1999. Ha estat guanyador en diverses ocasions dels premis de la comunitat *sourceforge* com a millors projectes per empresa i finalista com a millor projecte i millor projecte per governs. Actualment compta amb una comunitat d'usuaris àmplia i s'utilitza també en entorns de producció reals.

Com PostgreSQL, treballa en varis nuclis (Multi-Core) a nivell de connexió i Single-Core a nivell de consulta.

També és un sistema orientat a la transacció que compleix totes les propietats ACID i també utilitza el sistema de multiversions per assegurar una correcta concurrència.

El seu model d'emmagatzematge és NSM, on totes les files es guarden conjuntament en una mateixa pàgina de dades. Tot i així, és possible configurar la mida de pàgina que es vulgui utilitzar (1KB, 2KB, 4KB, 8KB i 16KB) però es recomana les més grans per a un millor rendiment.

Pel que fa a indexació, només es permet un únic tipus d'índex: els arbres balancejats ***B-Tree.***

Es disposa de dues versions: SuperServer i SuperClassic. En la primera es limita la *cache* de pàgines de dades a un màxim de 2 GB. En el segon, s'arrenca un procés per a cada connexió a la base de dades fent que el límit total de *cache* de pàgines sigui molt major. En aquest cas amb l'inconvenient de que no siguin compartides entre processos però amb l'avantatge de que aprofita millor el treball en entorns de multiprocessadors. En les dos versions no es té cap topall pel que fa a la mida de memòria dels algorismes d'ordenació (configurables per paràmetres).

Els algorismes de *join* disponibles són "*nested loop join*" i "*merge join*" i els d'ordenació "*quicksort*" si es pot realitzar en la memòria o "*external merge sort"* al disc.

Permet també claus primàries, claus foranies, claus úniques, *triggers*, *checks* i procediments emmagatzemats. [3]

2.5.4 Sistemes orientats a presa de decisió.

L'actual necessitat d'investigar en la informació per fer anàlisis, llistats i buscar tendències o patrons fa que apareguin noves aplicacions sobre les bases de dades. Així doncs, anomenarem OLAP (processament analític en línia) a aquest sector d'aplicacions que s'utilitzen per fer consultes complexes fetes al moment per satisfer necessitats puntuals (ad-hoc querys). L'objectiu principal del sistema ha de ser la velocitat de resposta.

Es comú executar les consultes OLAP en un copia separada de la base de dades principal, anomenada *data warehouse.* En aquesta s'hi poden unir dades provinents de diferents procedències, solen ser actualitzades en horaris no operatius i s'utilitzen únicament per fer lectures de dades (anàlisis). D'aquesta manera s'hi poden realitzar consultes complexes sense afectar el rendiment de la base de dades operativa que realitzarà tant lectures com escriptures, i que no podria suportar la càrrega de treball que aquestes consultes provocarien.

Actualment existeixen solucions diferents per poder satisfer les consultes OLAP. Des de sistemes que es distancien molt de les bases de dades relacionals (cubs multidimensionals) a sistemes relacionals orientats a presa de decisió. Els primers estructuren les dades en forma multidimensional i precalculen agregacions sobre una estructura d'estrella, descrit en la secció 2.5.5. Els segons son SGDB que s'implementen tenint en compte els requeriments de nomes lectura i velocitat. [4]

2.5.5 Esquema d'estrella.

Un esquema d'estrella es aquell que s'organitza en una taula principal anomenada *taula de fets* que s'uneix a moltes altres, anomenades *taules de dimensió*. La taula de fets, que sol ser la que més dades conté, actua com el centre de l'estrella i les de dimensió actuen com les seves puntes. Normalment la taula de fets disposa d'atributs que serviran per unir-se amb les dimensions i altres atributs que són dependents d'aquestes dimensions i que seran les dades d'importància a consultar. Es pot veure un exemple en la següent figura 2.8, on les vendes són la taula de fets i el temps, el client i el producte serien les taules de dimensió. La quantitat venuda es l'atribut dependent i els altres són tots atributs de dimensió lligats a les seves corresponents taules.

Aquesta estructura permet als cubs multidimensionals, precalcular valors basats en agregacions. Això es degut a que, per exemple, si es disposa de les quantitats venudes en un dia, es pot precalcular les vendes de la setmana, mes, trimestre i any. Si aquest concepte s'extén a subfamilies de productes o a poblacions, comarques i països de

clients, es podran obtenir dades molt ràpidament ja que no s'hauran de realitzar els càlculs en temps d'execució. D'aquesta forma, aquest concepte es replica a totes les dimensions on es considera que la seva agrupació serà freqüent en les consultes.

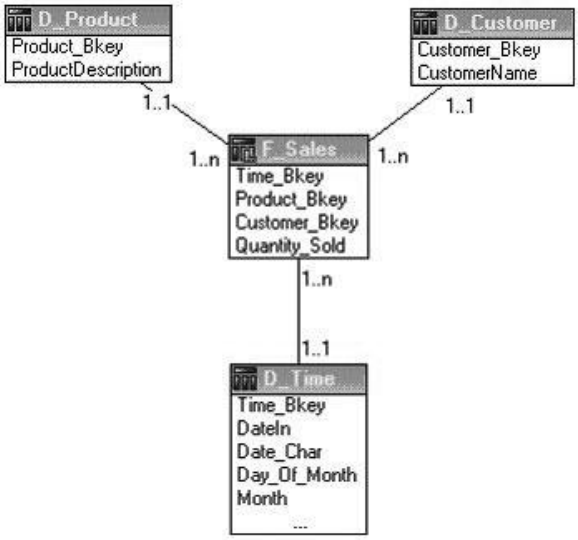


Figura 2.8: Exemple d'una esquema d'estrella

2.5.6 MonetDB

MonetDB es un sistema de bases de dades de codi lliure i multiplataforma. Apareix el 2004 com a resultat dels projectes d'investigació de l'universitat d'Amsterdam. Al 2008 es crea una branca privada de MonetDB anomenada VectorWise, que va ser comprada per Ingrés el 2011. Actualment VectorWise esta situada en els primers llocs del rànquing de bases de dades més rapides del món segons Consell de Processament de Transaccions sobre el seu joc de proves TPC-H.

El seu esquema d'execució de consultes es Multi-Core. Segons [5], s'utilitza l'esquema Map-Reduce.

Es un sistema orientat a la presa de decisions, aplicat en aplicacions OLAP d'altes prestacions.

L'emmagatzematge de dades es realitza de forma fragmentada per columnes (DSM). En aquest, cada columna de la base de dades es vectoritzada i emmagatzemada a disc de forma comprimida. Durant l'execució les dades es porten a memòria i es descomprimeixen només quan s'ha d'utilitzar per la CPU. D'aquesta forma aconsegueix aprofitar millor l'espai de memòria i permet portar un major nombre de pagines de dades.

No permet la utilització de cap mena d'índexs per part dels usuaris. D'altre banda, ella mateixa crea els índexs que creu convenient durant l'execució repetitiva de consultes. El tipus d'índex que crea són *hash*.

És una base de dades basada "in-memory" i no es pot configurar la mida de memòria que es podrà utilitzar. Com a recomanació es demana ampliar la mida de la memòria virtual del sistema operatiu per evitar errors.

De les anteriors, MonetDB es la més complexa a nivell d'algorismes utilitzats. D'ells es busca que siguin eficients en cache de CPU i s'han codificat específicament pel sistema. Segons [6], disposa de 335 operacions de join i pel que fa a l'ordenació, només podem dir que utilitza algorismes propers al *radix sort* (Radix Cluster).

Permet també, claus primàries, claus foranies, claus úniques , *triggers, checks* i procediments emmagatzemats.

2.6 Entorns Many-Core

A continuació es farà una introducció a la tecnologia GPU d'Nvidia, explicant la seva arquitectura i forma d'execució. Tot seguit s'exposen els motius pels quals pot ser adaptada pels sistemes de bases de dades per accelerar els temps de resposta de consultes.

2.6.1 GPUs i CUDA.

Nvidia ha impulsat els entorns *manycore* a través dels seus productes de computació per a GPUs (***G**raphics **P**rocessing **U**nit*). La utilització del llenguatge de programació CUDA (***C**ompute **U**nified **D**evice **A**rchitecture),* desenvolupat per la mateixa empresa, permet la programació altament paral·lela en aquests entorns, on s'aconsegueixen una alta potencia de computació i uns amples de banda a memòria elevats.

Aquesta potencia de càlcul fa que es pugui considerar una alternativa viable sobre certes aplicacions que s'adapten bé a les característiques dels entorns *manycore*, ja que poden donar més rendiment en temps i amb un consum baix d'energia.

D'altre banda, podem dir que les limitacions principals que presenten les GPU son la mida de la memòria principal (device memory), que sol ser més reduïda que la disponible en CPU tot i que més rapida. L'altre és el temps de transferència entre el host i la GPU a través dels ports PCIe, que provoca que al moure dades freqüentment entre la CPU i la targeta introdueixi un overhead.

El model d'execució utilitzat és el que Nvidia anomena SIMT (Single Instruction, Multiple Thread); Tots els cores d'un mateix grup executen la mateixa instrucció al mateix moment, semblant als processadors SIMD. Cal dir que en les instruccions condicionals, on es divergeix en l'execució, els cores del mateix grup s'habiliten i es deshabiliten per poder executar només el codi que els toca.

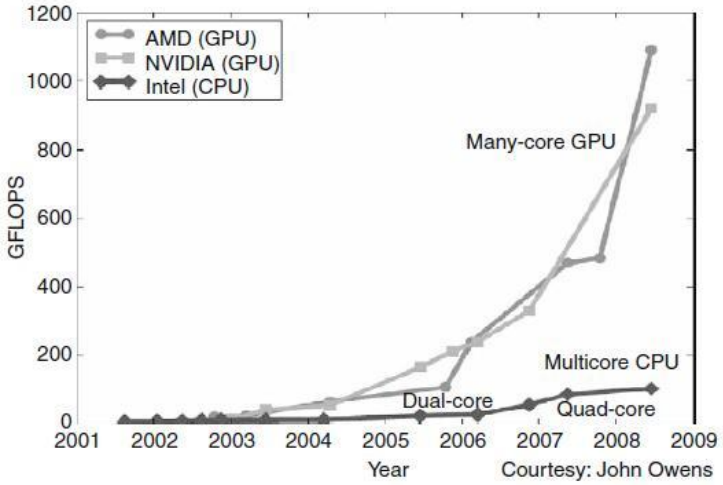


Figura 2.9 : Tendències en capacitat computacionals GFLOP/s de les diferents arquitectures.

La forma en que s'estructura l'execució d'un kernel (codi executable en GPU) és la següent; Tots els threads que es generen en la crida d'un kernel s'anomena *grid*. Aquest grid a la vegada es divideix en diferents *blocs de threads*, independents entre ells. Això implica que els threads situats en diferents blocs no es podran coordinar ni sincronitzar. Finalment, aquests blocs es divideixen en *warps* (32 threads), que es una limitació física i no modificable. Els warps no són una especificació de CUDA i al programador no li cal ser-ne conscient.

L'arquitectura de la GPU s'organitza en una matriu de SM (Stream Multiprocessors) capaços d'executar un alt nombre de threads amb un cost molt petit de creació i destrucció. Aquests SM estan composats de múltiples SP (Stream processors) que comparteixen lògica de control i cache d'instruccions. A més a més, també comparteixen mòduls d'operacions especials (sinus, cosinus, etc...) SFU, registres, un mòdul configurable de memòria compartida i cache L1 (48KB/16KB) i diversos mòduls load/store a memòria compartida/cache/DRAM.

Cada SM disposa dels seus propis mòduls de scheduling, dos en l'arquitectura anomenada Fermi. Aquests, son els encarregats de planificar l'execució dels warps dins dels SM depenent de la disponibilitat dels recursos. Tot i així, es pot dir que gràcies l'enorme quantitat de threads que s'esperen per ser executats degut a la paral·lelitzacio, quan un warp necessita un recurs no disponible, es canvia per una altre warp al següent cicle, fent que la latència de recerca a la memòria quedi amagada. Aquesta arquitectura es pot veure en la següent esquema (Figura 2.10)

Pel que fa a la jerarquia de memòria, es disposa d'una cache L2, compartida per tots els SM, i una memòria global. Aquesta DRAM es del tipus GDDR5 , disposa d'una ample de banda que pot superar els 100 GB/s i on es realitza l'intercanvi d'informació entre la GPU i la CPU a través del port PCIe. La representació física i l'esquema de l'arquitectura de memòria es poden veure en la següent figura 2.11. [7] i [8]

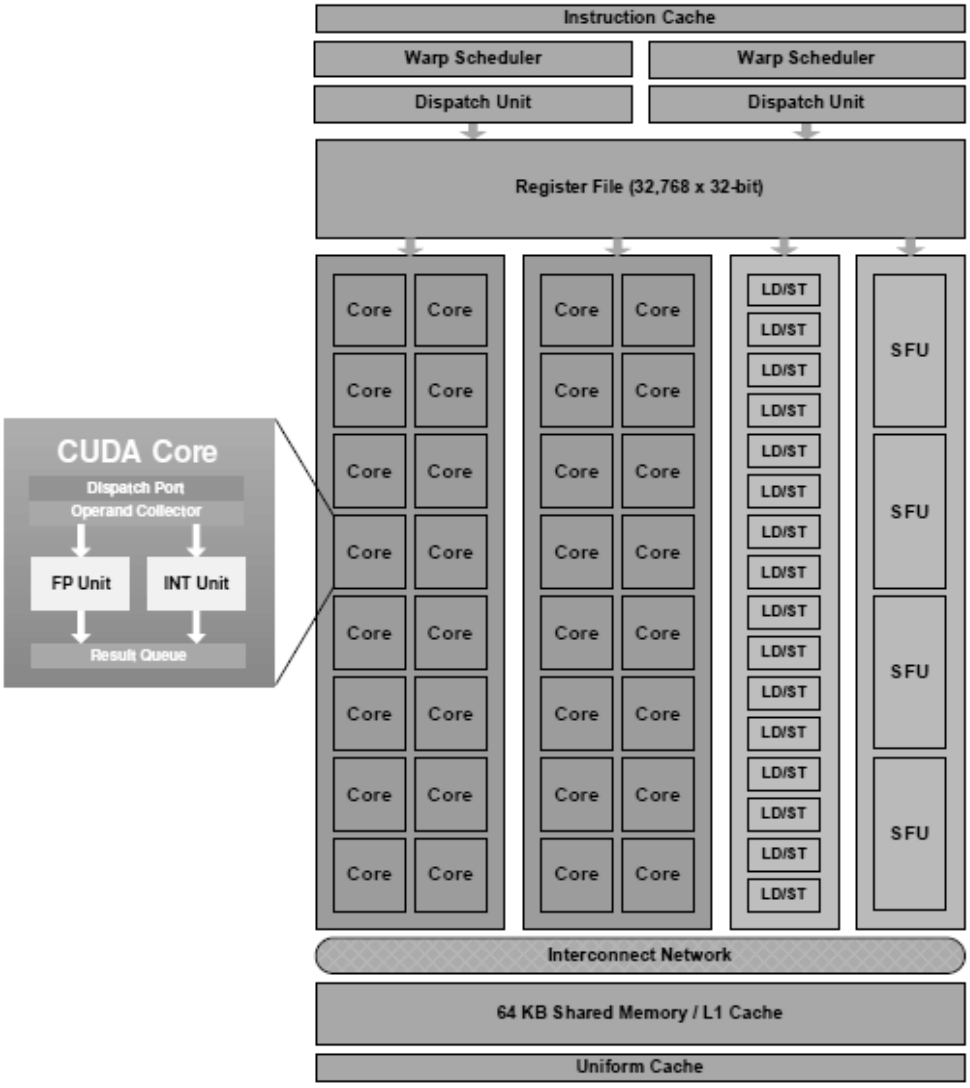


Figura 2.10 : Arquitectura d'una SM

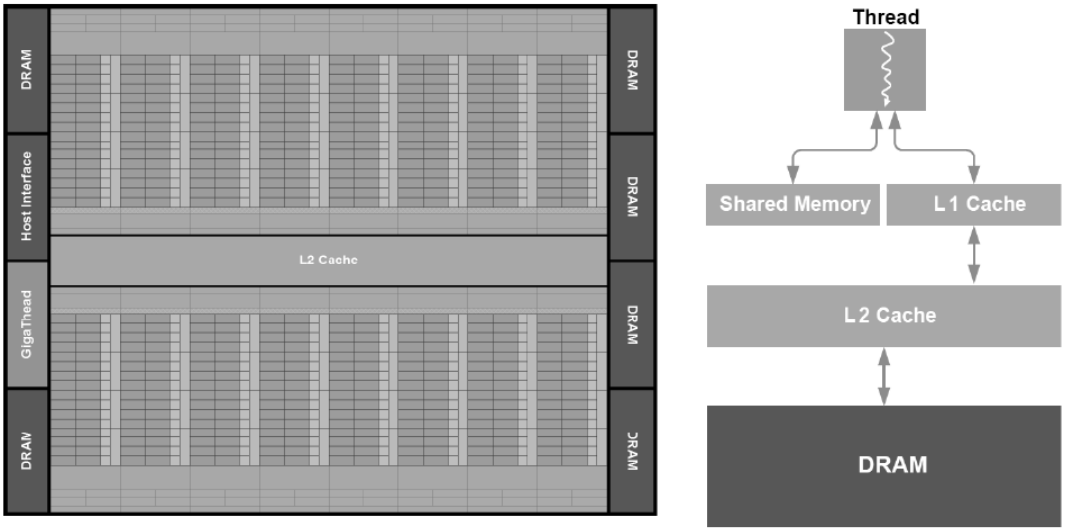


Figura 2.11 : A l'esquerra, l'esquema de l'arquitectura d'una GPU Fermi, on es poden veure els diferents mòduls. A la dreta, l'esquema de la jerarquia de memòria.

2.6.2 Bases de dades en entorns Many-Core

El subconjunt de sistemes de bases de dades destinades a presa de decisions es poden considerar aptes pel domini de les GPUs. Així doncs, al tractar-se principalment d'aplicacions de sols lectura de gran quantitat de dades fa que s'hagin iniciat projectes de recerca per intentar aconseguir accelerar els algorismes multi-core actuals.

Per altra banda, en entorns on podem vectoritzar verticalment les taules, com pot ser en sistemes que usen la capa DSM per emmagatzemar dades, podrem adaptar-nos fàcilment a les capacitats fonamentals de les GPU on el càlcul vectorial en paral·lel obté els majors guanys.

Addicionalment, per intentar solucionar el temps de transferència, es pot fer ús d'algorismes de compressió de dades. Partint d'unes pagines de dades de disc comprimides, reduint considerablement les operacions d'E/S a disc, es podrà enviar la informació comprimida a la targeta i descomprimir-les en temps d'execució. Algun d'aquests algorismes lightweight, que tenen ràpida descompressió i una taxa de comprensió acceptable serà avaluat per separat per considerar-ne la seva utilitat.

Pel que fa al sistema de bases de dades, s'ha obtingut el codi font d'un projecte de codi obert Alenka , que descriurem a continuació.

2.6.3 Alenka

Alenka, es un SGBD de recent creació, que permet l'execució d'un cert subconjunt de consultes TPCH per poder avaluar-ne el seu rendiment. A més, es capaç de realitzar operacions sobre conjunts de dades que no poden cabre completament en memòria de Host (CPU) ni en la de la GPU, obtenint altes prestacions.

Permet tot tipus de recerca de informació condicional, agregacions (sum, avg , count, etc...) i ordenacions de conjunts. Utilitza la llibreria Thrust, recent incorporada al SDK de Nvidia, que implementa operacions primitives sobre vectors per poder realitzar totes aquestes tasques.

Tot i això, degut a que en el moment en que escric aquesta memòria, el projecte té 7 mesos de vida, no s'ha implementa cap tipus de generador de pla de consultes ni optimitzador. D'aquesta forma l'usuari li ha de proporcionar, a través d'un llenguatge adaptat de l'SQL, un pla d'execució a través de descompondre una query en petites operacions tal com farien aquests mòduls no implementats.

La implementació de la capa d'emmagatzement també es primària. No disposa de cap estructura de *metadades* per poder associar un nom de relació a un fitxer. Per tant, sempre que es vulgui llegir una taula, s'haurà d'indicar el nom de fitxer on localitzar-les. Aquestes, es poden guardar en format pla o opcionalment binaritzades per facilitar i agilitzar la lectura. Els fitxers representen els atributs en forma vectorial, semblant a una DSM, per poder utilitzar només els atributs on realment s'han de fer les operacions.

Per la forma en que s'ha implementat, s'adapta bé sobre el benchmark TPCH, ja que aquest s'estructura en forma d'estrella on una taula amb un gran volum de dades es relaciona amb taules més petites. D'aquesta forma s'ha decidit que el sistema mantindrà en memòria el total dels atributs d'aquestes taules petites i iterativament portarà fragments de dades de la taula gran. En cada iteració es realitzen les operacions pertinents i s'actualitzen els resultats parcials. Dit això, s'ha de dir que només permet portar fragments a memòria d'una sola taula, que l'usuari podrà escollir. Per tant si el bechmark tingués dos taules grans que no cabessin a la memòria, no es podria fer cap operació.

Pel que fa a la memòria de la GPU, es mantindrà sempre lliure en un 50%. Això es degut a que el nombre de resultats de qualsevol operació sempre pot ser inferior o igual al nombre de registres dels conjunts d'entrada. Per tant, si els conjunts d'entrada caben al 50% restant sempre es podrà executar. Un cop finalitzat, s'allibera la memòria de la targeta i es transporten els resultats cap al Host.

També te limitacions pel que fa a formats de dades. Es possible acceptar diferents dominis de dades, però en codi seran representats com a integers, doubles o strings. Tot i així, s'ha de dir que permetre la utilització de strings de mida fixada ja es una gran utilitat, ja que els altres SGBD vistos sobre GPU en recerca no ho permetien.

Per últim, no com a limitació d'arquitectura, sinó també per falta de temps del programador, no s'hi ha implementat operacions SQL com ara el CASE o NONEQUI JOIN entre altres. Pel que reduirà el nombre de consultes que es podran executar.

Així doncs, utilitzaré aquest codi per poder avaluar des de la consulta 1 fins la 6 del benchmark TPCH amb diferents escalats per poder veure els seus comportaments. [9]

2.6.4 Primitives paral·leles

En aquest apartat s'exposen les primitives paral·leles disponibles en la llibreria de Cuda, Thrust. La majoria d'operacions que posen a disposició son altament paral·lelitzable i per tant s'adaptaran be en GPUs. Posteriorment, en el punt 5.1 s'explica com s'adapten les operacions SQL d'Alenka a traves d'aquestes primitives.

Transform

Algorisme que aplica una operació binaria a cada parella d'elements dels vectors d'entrada (amb la possibilitat de limitar-les a un subconjunt del vector).

En la Figura 2.12 es veu un exemple on es mostra el seu funcionament.

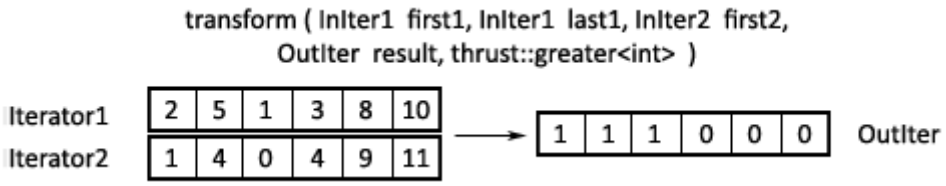


Figura 2.12 : Exemple del funcionament de l'operació *transform* usant l'operació binaria *greater<int>*.

Gather

Algorisme que reodena els elements d'un vector a partir dels valors d'un segon vector que fa el mappeig. Aquest segon vector indica a on s'han d'anar a buscar els valors pel vector de sortida (sortida[i] = entrada[mapeig[i]])

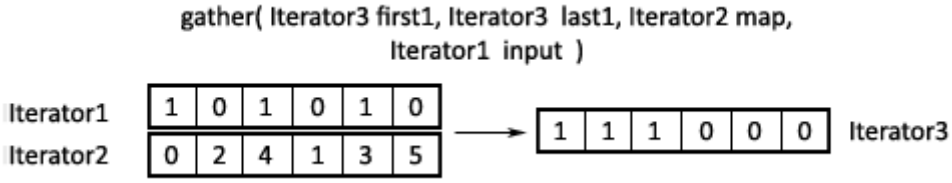


Figura 2.13: Exemple del funcionament de l'operació *gather*.

Scatter

Algorisme que reodena els elements d'un vector a partir dels valors d'un segon vector que fa el mappeig. Aquest segon vector indica a on s'han de guardar els valors del vector de valors dins el vector de sortida (sortida[mapteig[i]] = entrada[i])

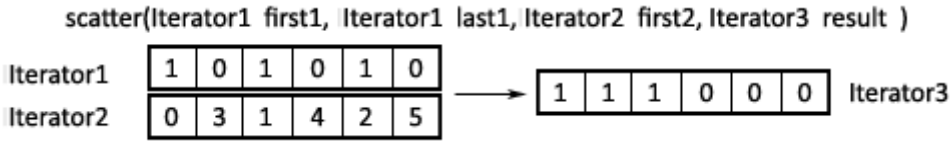


Figura 2.14 : Exemple del funcionament de l'operació *scatter*.

Copy if

Algorisme que copia els elements a un vector de sortida si els elements amb mateix índex compleixen una certa condició.

En la Figura 2.15 es veu un exemple on es mostra el seu funcionament.

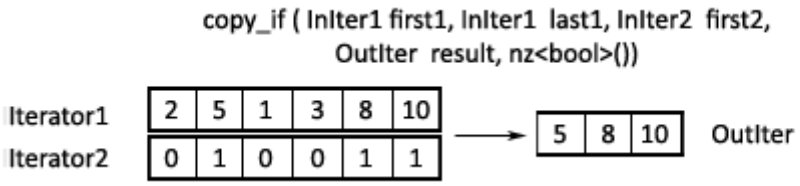


Figura 2.15 : Exemple del funcionament de l'operació *copy_if* usant la condició *nz<bool>* (diferent de zero).

Scan

Algorisme que aplica una operació unària a tot un vector d'entrada. Existeix la versió inclusiva, on el valor actual es considerat en el resultat, i la versió exclusiva, on el valor actual no es considera en el resultat.

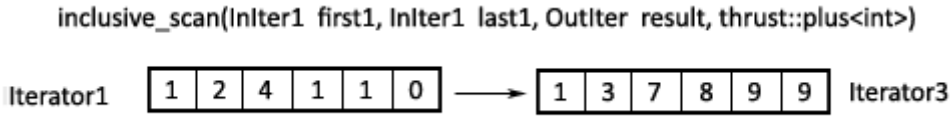


Figura 2.16 : Exemple del funcionament de l'operació *inclusive_scan* usant l'operació binaria *plus<int>* (suma).

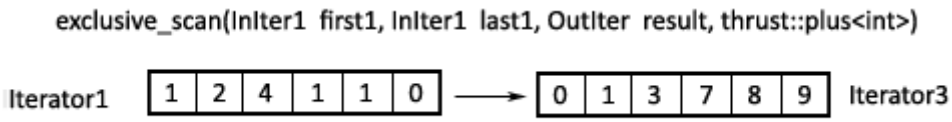


Figura 2.17 : Exemple del funcionament de l'operació *exclusive_scan* usant l'operació binaria *plus<int>* (suma).

També es pot utilitzar les versions amb key, on es realitza el mateix procediment però es segmenta per claus.

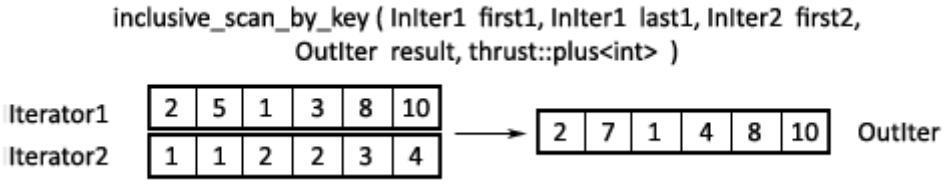


Figura 2.18 : Exemple del funcionament de les operacions *inclusive_scan* _by_*key* usant l'operació binaria *plus<int>* (suma), i on *Iterator2* actua com a *keys* per la segmentació.

Reduce

Algorisme que redueix tot un vector a un resultat final a partir d'una operació unària.

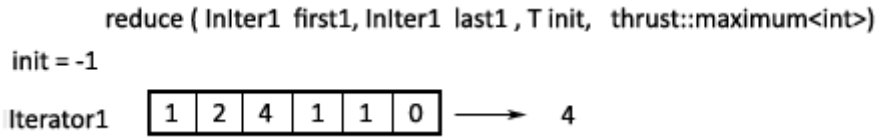


Figura 2.19 : Exemple del funcionament de l'operació *reduce* usant l'operació unària *maximum*<*int*> i inicialitzant a -1.

Sort

Algorisme que ordena tot un vector a partir d'una condició. Permet també ordenar un vector a partir d'un altre de mateixa longitud i que actua de keys.

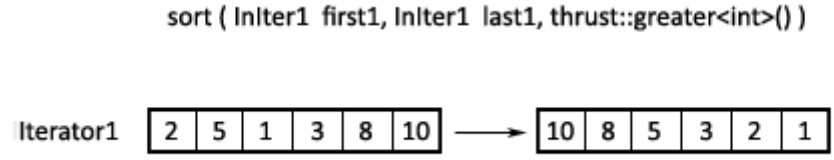


Figura 2.20 : Exemple del funcionament de l'operació *sort* usant l'operació unària *greater*<*int*>.

Es possible usar ordanació "stable" on si dos elements tenen el mateix valor es manté l'ordre.

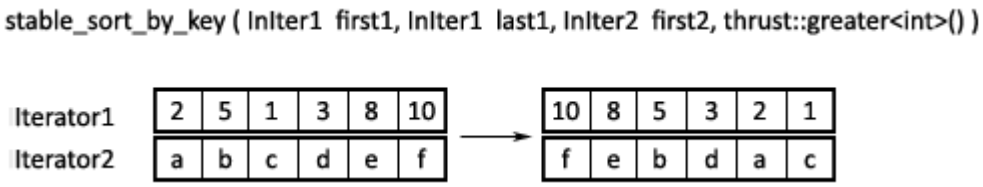


Figura 2.21 : Exemple del funcionament de l'operació *stable_sort_by_key* usant l'operació unària *greater*<*int*> i on *Iterator2* actua de *keys*.

Sequence

Algorisme que inicialitza un vector amb valors que s'incrementen per cada posició.

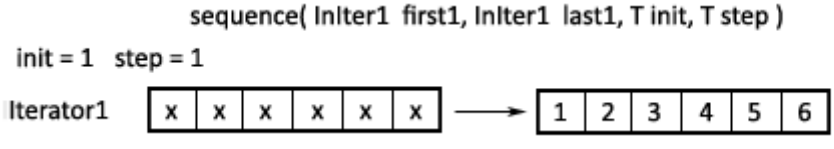


Figura 2.22 : Exemple del funcionament de l'operació *sequence* inicialitzada a 1 i amb un increment de 1.

Lower Bound i Upper Bound

Algorisme que aplica una cerca binaria vectorial contra un altre vector, un d'ells prèviament ordenat. En lower Bound ens retorna per cada valor del primer vector, l'index més petit on podria insertar-se del segon vector i que permetria a aquest seguir mantenint l'ordre. El Upper Bound, fa el mateix però retorna l'índex més gran que permet mantenir l'ordre.

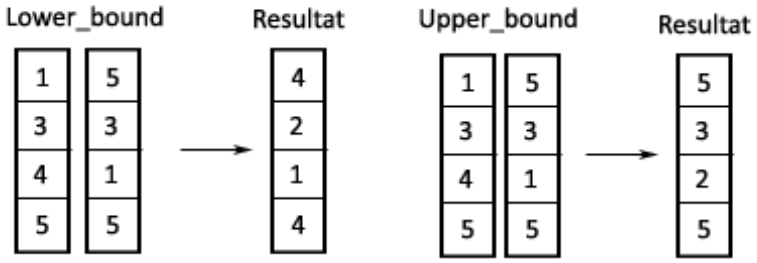


Figura 2.23 : Exemple del funcionament de les operacions *Lower_bound i Upper_bound.*

3 Metodologies de profiling i eines d'avaluació

En aquest apartat es troben les arquitectures i hardware utilitzat així com les diferents utilitats que s'han fet servir per obtenir els resultats. D'aquestes les separarem en les utilitzades per realitzar perfils d'aplicació i comprovar el rendiment, i unes altres que s'han utilitzat per aconseguir crear les bases de dades i obtenir els temps de resposta.

3.1 Arquitectura i màquines utilitzades

A continuació es mostra les arquitectures CPU i GPU utilitzades per executar el benchmark d'aquest projecte.

Intel Core 2 Quad

És l'arquitectura utilitzada en tots els jocs de probes. Disposem de 4 nuclis amb una *cache* L1 de 32 Kb per cada nucli. Aquests comparteixen, cada dos, una L2 de 6MB (Figura 3.1).

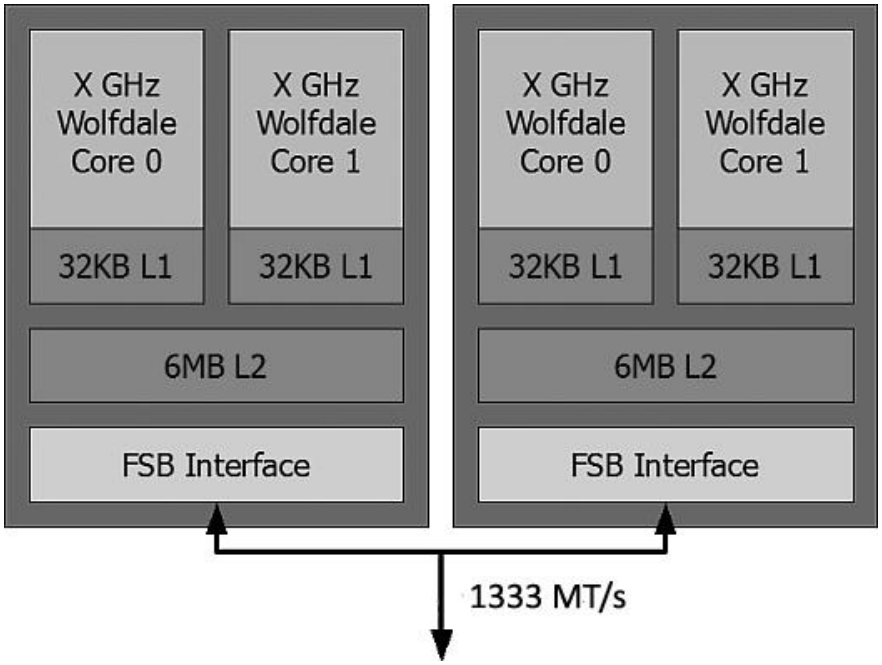


Figura 3.1 : Arquitectura Core 2 Quad

Les característiques especifiques de la maquina utilitzada són:

Intel Core2 Quad Q9550	
Cores	4
Threads	4
Rellotge de processador	2.83 Ghz
Cache L2	12MB
Bus/Core Ratio	8.5

5 GB de memòria 800MHz DDR3 Non-ECC

Sistema operatiu Windows XP Professional 64 bits
Disc dur de 250 GB per sistema operatiu i aplicacions
Disc dur de 1 TB per a dades

GPU

Per a la implementació en GPU s'ha utilitzat la targeta de Nvida GeForce GTS 450. Es basa en la nova arquitectura Fermi on s'incorpora una nova jerarquia de cache L1 i L2 i un segon WARP scheduler per a cada SM, entre altres millores. Les característiques especifiques de la targeta son les següents:

Nuclis de processament:	192
Rellotge de processadors:	1566 MHz
Memòria estàndard:	1 GB GDDR5
Rellotge de memòria:	1804.00 MHz
Amplada del bus de memòria:	128-bits
Ample de banda de memòria:	57.7 GB/s
Mida de Cache L2:	2
Mida de memòria costant:	65536 bytes
Mida de memòria compartida per bloc:	49152 bytes
Nombre de registres disponibles per bloc:	32768
Nombre màxim de threads per bloc:	1024

3.2 Eines de profiling

3.2.1 AQTime

AQTime es una aplicació comercial per fer anàlisis de rendiment basat en temps i events d'aplicacions CPU x86 i x64. Disponible en Windows. S'ha utilitzat la versió 7.0 en format d'avaluació.

D'aquest entorn, l'eina que s'ha utilitzat principalment es el Call Graph. D'aquesta forma s'ha pogut obtenir el percentatge del temps total que realitza cada operació interna. S'ha de dir però que en alguns casos els temps obtingut aplicant el Call Graph ha incorporat un alt overhead, causat suposadament per a la pròpia instrumentalització. Per tant aquests temps no s'han utilitzat i només s'han fet servir els percentatges fent les comprovacions pertinents per poder-los validar.

3.2.2 NVIDIA Visual Profiler

Es l'eina d'anàlisis de rendiment que ha creat Nvidia per poder estudiar el comportament de les aplicacions creades per GPU en Cuda C/C++ i OpenCL. Es compatible amb totes les targetes GPU posteriors al 2006 en Linux, Mac/OS i Windows.

Per Windows, es disposa d'una nova versió anomenada Nvidia Parallel Nsight però nomes per versions posteriors a Windows Vista i limitat en funcionalitat si no es disposa d'una segona targeta GPU.

S'ha usat per comprovar els funcionament dels kernels en memòria GPU.

3.3 Eines de base de dades

3.3.1 Benchmark TPCH

TPC Benchmark H (TPCH) [10] es un benchmark estandarditzat per el consell de processament de transaccions. En aquet es realitzen 22 consultes de propòsit específic (ad-hoc querys) i orientades a les aplicacions de presa de decisions sobre un gran volum de dades. Simula una base de dades real sobre una empresa majorista de ventes, i les consultes han estat dissenyades perquè no es pugui aprofitar cap optimització feta a priori.

És un joc de probes que ha de permetre comprovar el rendiment dels SGBD sobre aquest tipus de consultes analítiques, i per tant ressaltar les diferencies que apareixen entre els models tradicionals i els models que s'han especialitzat en aconseguir un temps de resposta petit sobre qualsevol tipus de consultes.

Pel que fa a la mètriques que s'especifiquen en el joc de probes, es disposa de dos maneres d'avaluar-les. En primer lloc es pot realitzar la mètrica de rendiment que es mesura en nombre de consultes finalitzades per hora (QpH). En segon lloc, es permet expressar els resultats en potencia / nombre de consultes finalitzades per hora (Watts/(K*QpH)). De totes maneres la mètrica que utilitzarem en el projecte es la de temps per consulta, que es relaciona directament amb la mètrica de rendiment especificada, però evita haver d'avaluar consultes concurrents. Això es degut a que en GPU només es possible executar una sola consulta al mateix moment.

L'esquema de les bases de dades s'especifica de la següent forma, on SF es el factor d'escalat de base de dades:

LINEITEM: El nombre de registres que conté es SF*6,000,000. Representen les línies de comanda.

ORDERS: El nombre de registres que conté es SF*1,500,000. Representen els comandes.

PARTSUPP: El nombre de registres que conté es SF*800,000. Representen proveïdors d'articles.

PART: El nombre de registres que conté es SF*200,000. Representen articles.

CUSTOMER: El nombre de registres que conté es SF*150,000. Representen clients.

SUPPLIER: El nombre de registres que conté es SF*10,000. Representen proveïdors.

NATION: El nombre de registres que conté es 25. Representen països.

REGION: El nombre de registres que conté es 5. Representen continents.

I les relacions entre les taules es la següent:

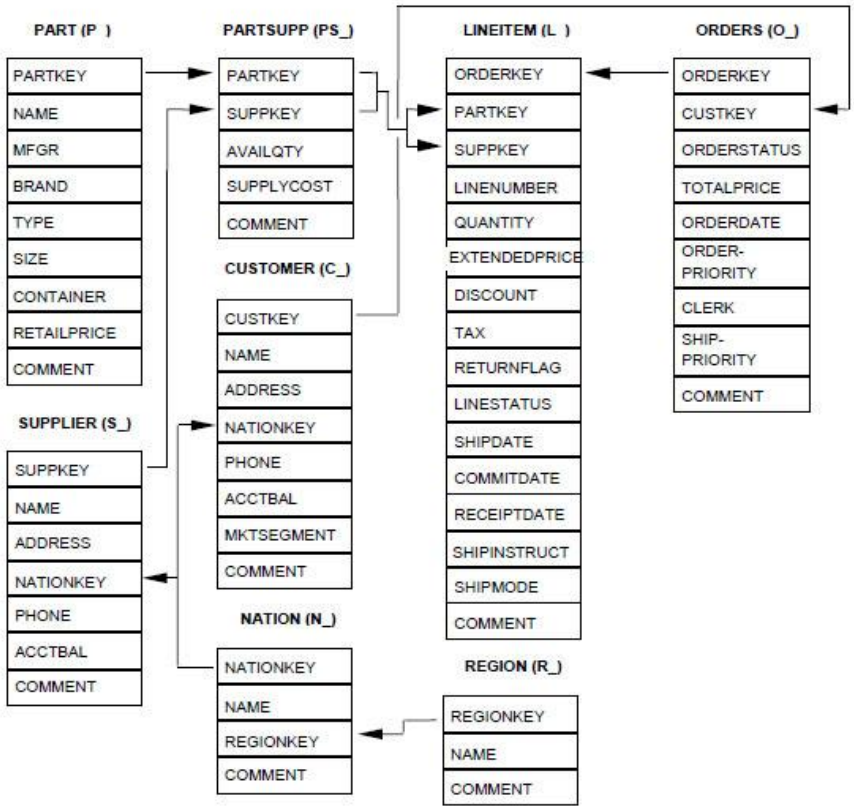


Figura 3.2 : Esquema i relacions de la base de dades del TPCH.

3.3.2 DBgen

Es un generador de dades estructurades per als jocs de probes de TPC. En Windows ve compilat en el paquet de TPCH, junt amb el fitxer de definicions que s'utilitza per generar les dades. En aquest fitxer es defineixen els requeriments de les taules, com ara l'estructura i les paraules que poden ser utilitzades per als camps de descripcions. Els resultats obtinguts es un fitxer amb extensió TBL per cada taula.

Un dels paràmetres que s'ha d'indicar al moment d'execució és l'escalat (per defecte 1GB). Aquest fa que els resultats obtinguts siguin de la mida indicada seguint els requeriments de les taules.

Per generar les dades per una base de 5 GB s'ha d'executar el següent:

dbgen.exe -v -s 5

3.3.3 Fsql

Es una eina de línia de comandes per bases de dades Firebird. Exten les funcionalitats de l'eina estàndard, ja que incorpora millor representació dels resultats de com s'ha executat la consulta. S'utilitzarà per la creació de les bases de dades de Firebird a partir dels fitxers TBL anteriors, així com per obtenir els temps d'execució de les consultes. Permet llegir fitxers d'entrada i escriure resultats en un fitxer de sortida.

3.3.4 Psql

Es una eina molt semblant a Fsql pero per a bases de dades PostgreSQL. Es farà servir pel crear les bases de dades en PostgreSQL i obtenir els seus temps d'execució de consultes.

3.3.5 Mclient

Versió per MonetDB de les eines anteriors. De les tres, la menys adaptada per a l'obtenció de resultats, ja que no permet obtenir els plans d'execució de les consultes, i per tants no es podrà saber la forma en que s'han executat. S'utilitzarà per a la creació de les bases de dades de MonetDB i el llançament de consultes.

4 Avaluació dels sistemes de bases de dades

En aquest apartat descrivim les configuracions dels diferents SGBD. A continuació s'indica el subgrup de consultes del joc de proves TPCH que s'utilitzaran. Finalment es mostren els resultats obtinguts pels diferents sistemes triats.

4.1 Configuració i inicialització dels SGBD

4.3.1 Configuració de PostgreSQL

PostgreSQL disposa de varies desenes de paràmetres que es poden modificar per adaptar-se als recursos disponibles. De tots ells, només s'han modificat els que afecten a la memòria que es pot utilitzar:

Effective cache size (128MB per defecte): Mida total de memòria RAM que es reservarà per la instancia del servei. Es recomana, en entorns dedicats, situar aquest valor a la meitat de la memòria real. En aquest projecte s'ha deixat a 2 GB.

Shared buffers (32MB per defecte): Mida del buffer utilitzat per guardar pagines de dades. Es l'espai on es realitzaran el treball de les consultes i es coordina amb la cache del sistema operatiu. S'ha situat a 512 MB.

Work mem (1MB per defecte): Espai que es reserva per operacions d'ordenació. S'ha situat a 512MB.

Random page cost (4 per defecte): Es un valor utilitzat en heurística per saber si es preveu anar mes vegades a disc o a memòria (valors grans indica anar més a disc). S'ha reduït a 2 ja que com que hem incrementat la memòria disponible, es preveu que hi hauran més accessos aleatoris, o el que es el mateix accessos a través de índex.

4.3.2 Configuració de Firebird

Tal com s'ha fet amb l'anterior cas, amb Firebird també s'han modificat els paràmetres que afecten a la memòria disponible:

Pàgina de dades: S'han creat les bases de dades amb una mida de 8KB.

DefaultDBCachePages (2048 per defecte): Nombre de pàgines de dades que es poden guardar en memòria RAM. S'ha situat al seu màxim, 131072 (1 GB).

TempDirectories: S'ha situat els directoris temporals a un disc amb força espai lliure. Això s'ha fet, ja que en cas de no poder treballar en memòria, utilitza tot l'espai de disc lliure disponible.

TempCacheLimit (64 MB per defecte): Mida de memòria que es pot utilitzar per realitzar operacions temporals com ara les ordenacions. S'ha situat a 1 GB.

TempBlockSize (1MB per defecte): Mida mínima de bloc que s'emmagatzemarà en l'espai temporal. S'ha situat a 128MB.

4.3.3 Configuració de MonetDB

En MonetDB no s'han realitzat canvis. Els únics requeriments que es demanen si es vol treballar amb bases de dades de mida gran és disposar de varies GB de memòria real i ampliar la memòria virtual del sistema operatiu (s'ha situat a 16 GB).

4.3.5 Configuració Alenka

Alenka és un projecte que s'ha iniciat recentment i no disposa de cap mena de configuració a través de paràmetres. Així doncs, en primer lloc es va realitzar un estudi del flux d'execució per poder entendre millor com treballa.

Com a resultat, es va aconseguir modificar la mida de bloc utilitzat en les iteracions realitzades durant la segmentació de la taula més gran. D'aquesta forma i llançant les consultes del joc de proves es va trobar un valor per obtenir un temps de resposta el més petit possible.

En les següents figures 4.1 i 4.2 es mostren les gràfiques de la suma de tots els temps de les consultes per cada mida de bloc, en milions de registres. D'aquesta manera es pot veure com a l'augmentar la mida de bloc el temps d'execució es redueix.

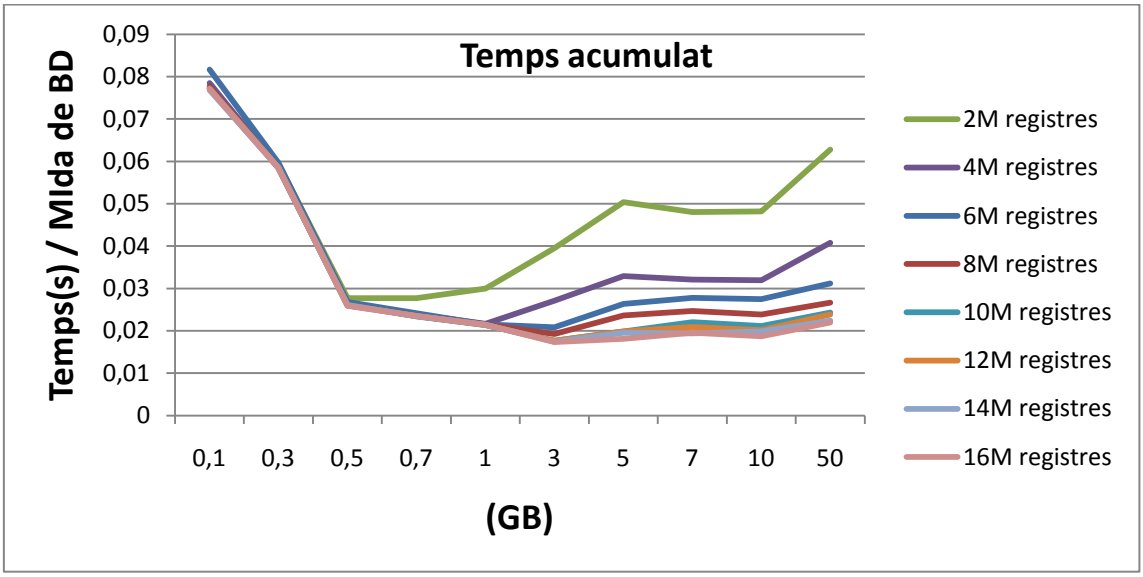


Figura 4.1 : Gràfica de la suma de temps de les 6 consultes realitzades en Alenka (GPU) per diferents mides de bloc (en milions de registres) i mides de BD. Escala en Temps(s)/ mida BD.

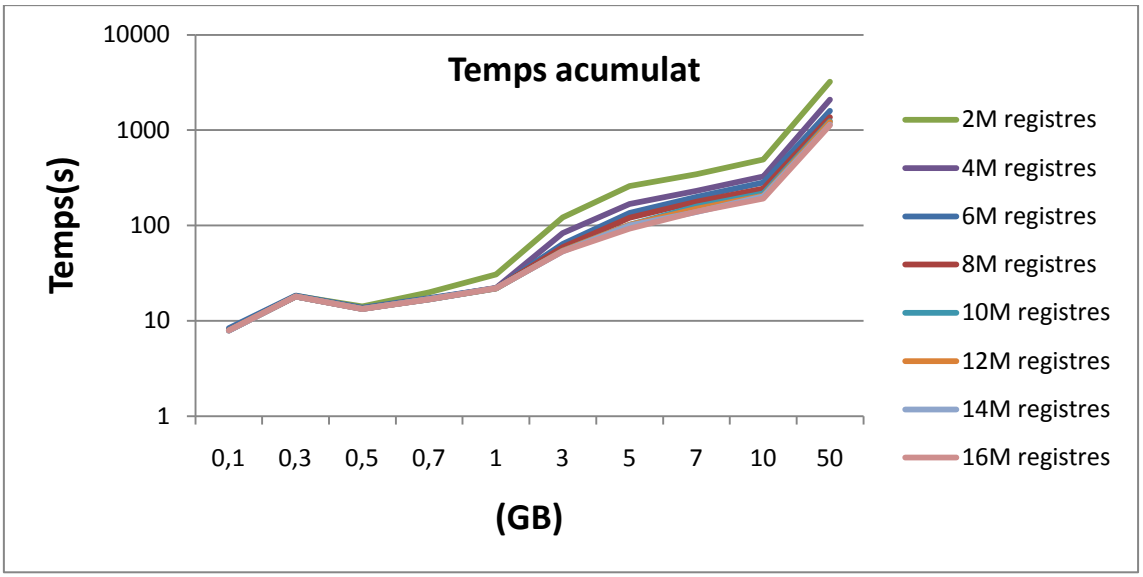


Figura 4.2 : Gràfica de la suma de temps de les 6 consultes realitzades en Alenka per diferents mides de bloc (en milions de registres) i per diferents mides de BD, en escala logarítmica.

Finalment s'ha escollit una mida de bloc de 16 milions per fer la comparació amb els altres sistemes.

Dit això, s'ha de dir que es pot continuar augmentant la mida de bloc i el temps de resposta segueix disminuint. Però es converteix en inestable i es produeixen errors de memòria en alguns casos, fet que m'ha decidit a no utilitzar-los amb el total de memòria RAM del que disposava.

4.2 Consultes utilitzades

A continuació es fa una breu introducció a les sis consultes utilitzades en el benchmark.

Q1 (Informe de resum de preus)

Proporciona un resum de preus per totes les línies de comandes que han estat enviades abans o igual a una determinada data. Ens retorna la suma i la mitja de les quantitats demanades, així com els preus i els descomptes, agrupats i ordenats per dos dels seus camps.

Aquesta consulta únicament tracta amb dades de la taula més gran (LINEITEMS) i s'exigeix que la data del filtre inclogui entre el 95% i el 97% dels registres perquè s'hagi d'escanejar la major part de les dades.

Q2 (Proveïdors amb preu mínim)

Troba quin proveïdor hauria de ser seleccionat per fer una comanda amb preu mínim en una certa regió per un cert article de mida i tipus determinat. D'aquests resultats ens proporciona la seves dades personals així com el nom del producte i la marca, ordenats per quatre dels seus camps.

Aquesta consulta exclou la taula LINEITEMS, i nomes treballa amb PART, SUPPLIER, PARTSUPP, NATION, REGION. D'aquestes PARTSUPP es la que més dades té. La resta realitzen equi-join per clau primària entre elles, i s'haurà de realitzar una subconsulta per poder trobar el preu mínim en el filtre

Q3 (Prioritat d'enviament)

Retorna les 10 comandes no enviades amb un valor major en una determinada data.

Aquesta consulta realitza equi-join per clau primària sobre les taules LINEITEMS, ORDERS i CUSTOMERS. Sobre aquests es filtra per una certa data per fer que la comanda s'hagi realitzat abans i l'enviament s'hagi fet després d'aquesta. Tot això s'agrupa i s'ordena per tres dels seus camps.

Q4 (Comprovació de prioritat de comanda)

Realitza una comprovació per analitzar el nombre de comandes que han estat rebudes posteriorment a la data en que es va comprometre amb el client. D'aquestes es separen per grau de prioritat que tenien, per poder analitzar el grau que funciona pitjor.

Aquesta consulta treballa sobre dos taules. En la primera, ORDERS, es mira en el rang d'un trimestre quina d'elles disposa d'articles enviats posteriorment de la data compromesa. Per fer-ho s'ha de buscar per cadascuna d'elles en la taula LINEITEMS si existeix algun registre que ho compleixi. S'ha de realitzar un equi-join entre elles, i recerques desiguals amb dates. Finalment l'ordena i agrupa pel camp de prioritat d'enviament.

Q5 (Volum per proveïdor local)

Retorna els ingressos aconseguits a traves de comandes proveïdors on els clients i els proveïdors son de la mateixa regió a un any vista sobre una data determinada.

Aquesta treballa sobre CUSTOMER, ORDERS, LINEITEM, SUPPLIER, NATION, REGION, totes elles unides per equi-joins. Es realitzen recerques desiguals per data. Tots els resultats s'agrupen i s'ordenen per dos camps diferents.

Q6 (Canvi de previsió d'ingressos)

Retorna els ingressos que hauríem aconseguit si haguéssim eliminat un cert rang de descomptes en un cert any determinat.

Aquesta consulta traballa unicament sobre LINEITEMS. Sobre ella realitza recerques desiguals de dates, i recerques entre intervals.

4.4 Resultats generals de rendiment

A continuació es mostren els resultats d'execució de les consultes escollides del benchmark TPC-H. Com a resum, en aquest apartat només es mostren els resultats en forma de mitja, on s'han sumat els temps de les sis consultes per cada mida de base de dades i per cada sistema de gestió de bases de dades. Els resultats específics de cada consulta poden ser consultats en l'Annex 8.1.

Les dos gràfiques (Figura 4.3 i 4.4) mostrades utilitzen escales diferents. En la primera gràfica es mostren els valors de temps (en segons) dividits per la mida total de la base de dades. Aquest mètode presenta una inexactitud, ja que el volum de dades usat pels algorismes de cada SGBD difereixen en cada cas segons el seu esquema d'emmagatzematge (alguns han de portar a la memòria RAM dades que no s'utilitzaran per processar). Tot i això, es considera que es una mètrica adequada des del punt de vista d'un usuari final, on pot comprovar els temps obtingut a partir de la seva mida total de base de dades.

Com es pot veure, en l'instant en que la mida del problema es fa gran es poden agrupar els SGBD en dos grups. Un el formen els sistemes orientats a la transacció (PostgreSQL i Firebird) i l'altre els sistemes orientats a l'anàlisi (MonetDB i Alenka).

En la segona gràfica, s'utilitza una escala logarítmica per poder mostrar de forma real l'evolució dels temps (en segons) obtinguts en cada cas.

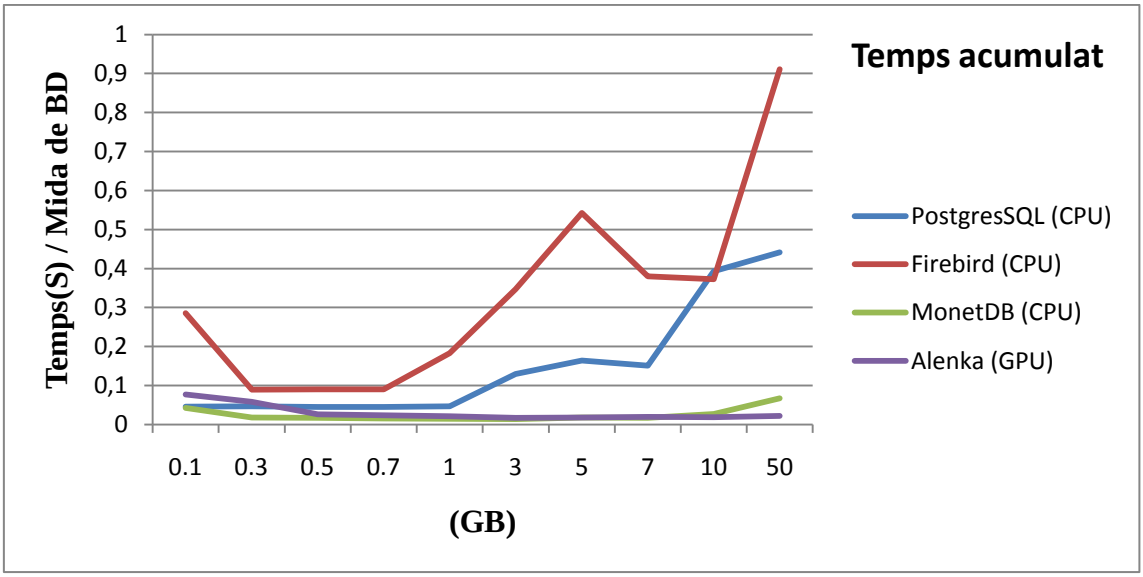


Figura 4.3 : Gràfica de la suma de temps de les 6 consultes realitzades en els diferents SGBD, per diferents mides de bases de dades. Escala en Temps(s) / mida BD.

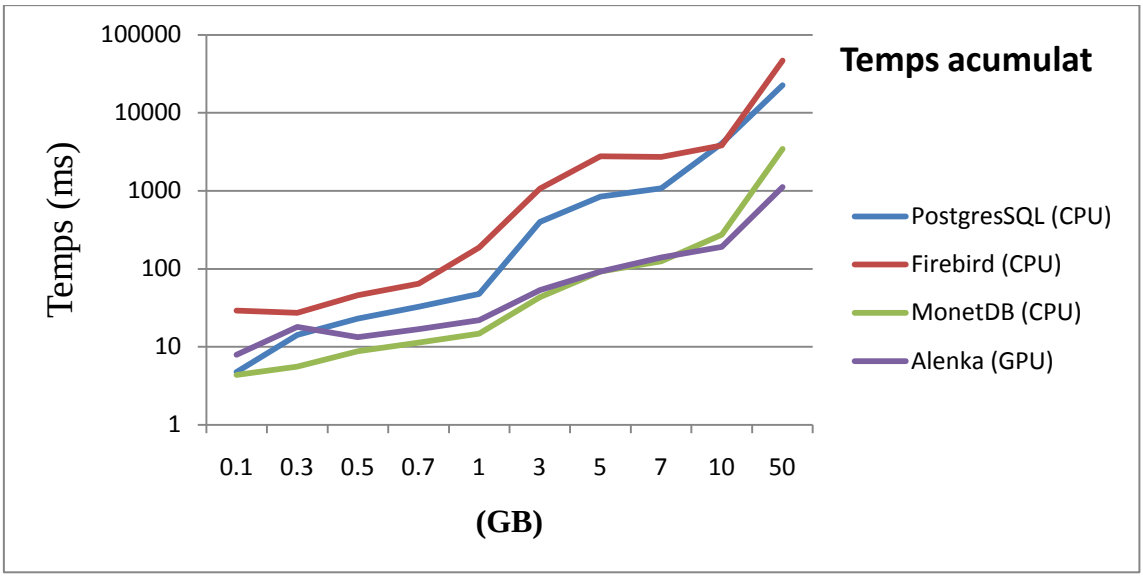


Figura 4.4 : Gràfica de la suma de temps de les 6 consultes realitzades en els diferents SGBD, per diferents mides de bases de dades. Escala logarítmica.

En primer lloc, cal recordar que els sistemes orientats a la transacció disposen de entre 1 i 2 GB de memòria RAM per realitzar les seves operacions (primitives SQL i ordenacions). En cas de que els conjunts entremitjos superin aquests valors, les hauran d'executar utilitzant el disc.

D'altre banda MonetDB és un sistema SGBD "*in-memory*" i no es pot limitar a utilitzar un determinat valor de memòria. Usarà memòria virtual en cas de que els seus conjunts superin els recursos disponibles.

Per últim, Alenka només llegeix de disc les columnes necessàries i les porta a memòria. La taula que més dades té, serà partida en blocs i portada a la memòria RAM a demanda, actualitzant els resultats entremitjos. Un cop s'ha llegit del disc, totes les operacions seran realitzades per la GPU i s'usarà la memòria RAM de la CPU per mantindre els conjunts útils durant l'execució. En cas de sobrepassar la memòria disponible no es podran retornar els resultats (cas que mai ha passat durant el benchmark).

Un cop situat en context, es comentaran els resultats agrupant els SGBD en dos grups. El primer, que es comentarà més en profunditat en aquest apartat, és el que formen els dos sistemes orientats a la transacció (PostgreSQL i Firebird). El segon, es comentarà al següent punt 5.2, ja que els seus rendiments quan els volums s'incrementa es diferencien molt clarament.

Així doncs, primerament es pot veure que mentre la mida del problema es manté per sota de 1 GB les diferencies entre els sistemes son ja observables però cap d'ells es degrada del seu rendiment esperat.

Dels dos sistemes orientats a la transacció, destaca PosgreSQL. Amb petits volums de dades i mentre pot realitzar accessos a dades optimitzades a través d'índexs els seu

rendiment es bo en pràcticament totes les consultes, tret de la Q1 on els SGBD orientats a decisió van fins a 10 vegades més ràpid.

L'altre sistema, Firebird, es el que pitjor rendiment ofereix. En les Q1, Q2 i Q3 els seus temps son 10 vegades més lents que el més ràpid (MonetDB). Tot i així es pot dir que en la Q4, Q5 i Q6 el seu rendiment es molt pròxim a PostgreSQL.

En el moment en que el volum de dades supera 1 GB, aquests dos SGBD es comencen a tornar inestables i es degraden. Es pot dir que en totes elles van de 20 a 100 vegades més lentes comparades amb Alenka i MonetDB segons la consulta. Tot i així, cal remarcar que PostgreSQL ofereix uns resultats millors, amb una degradació menys pronunciada i amb una tendència més previsible que permet tenir-la en compte en producció.

Per tant en aquestes situacions ja es poden descartar per realitzar aquest tipus de consultes. Els seus rendiments en temps no seran útils per aquests tipus de consultes ja que una de la necessitats que es demana en les aplicacions analítiques es la ràpida resposta. A partir d'aquest punt ja no es tindran en compte per poder fer una comparació més exacta de la implementació GPU.

4.5 Resultats detallats de rendiment

En aquest apartat s'han eliminat els valors dels dos SGBD orientats a transacció i s'han deixat els dos que més rendiment ofereixen.

A continuació es mostren les gràfiques en forma de mitja (Figures 4.5 i 4.6) seguides de totes les consultes utilitzades. S'usarà l'escala de temps (en segons) dividida per la mida total de la base dades i l'escala logarítmica (en milisegons).

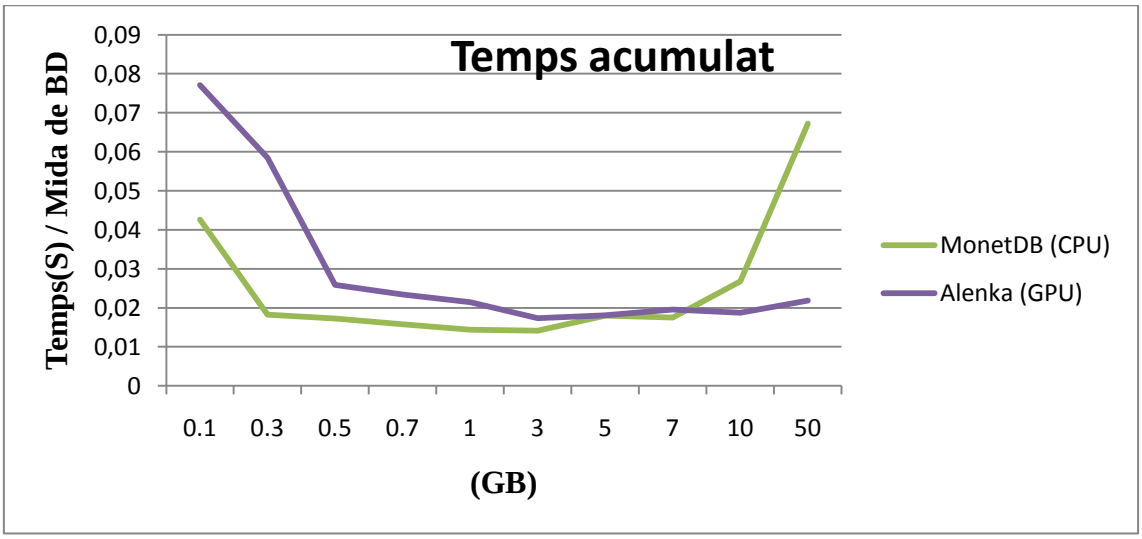


Figura 4.5 : Gràfica de la suma de temps de les 6 consultes descartant els dos sistemes orientats a la transacció, per diferents mides de bases de dades. Escala en Temps(s) / mida BD.

Com es pot veure, mentre les mides es mantenen per sota de 3GB, MonetDB presenta un millor rendiment. En els casos entre 0.1 i 1GB obtenen uns *speedups* d'un rang ampli de entre 1 i 30. Posteriorment (1 – 3 GB) es redueix a un *speedups* de entre 1 i 3.

Tot seguit, entre les 3 i 7 GB , els dos sistemes s'equilibren però ja es presenta una tendència en que es veu com Alenka comença a superar al seu competidor.

Entre 7 i 50 GB MonetDB es degrada considerablement mentre que Alenka mantén bastant bé la tendència anterior traçant una corba pràcticament lineal. El *speedup* obtingut és de entre 1,5 i 3 vegades més ràpid, en aquest últim rang.

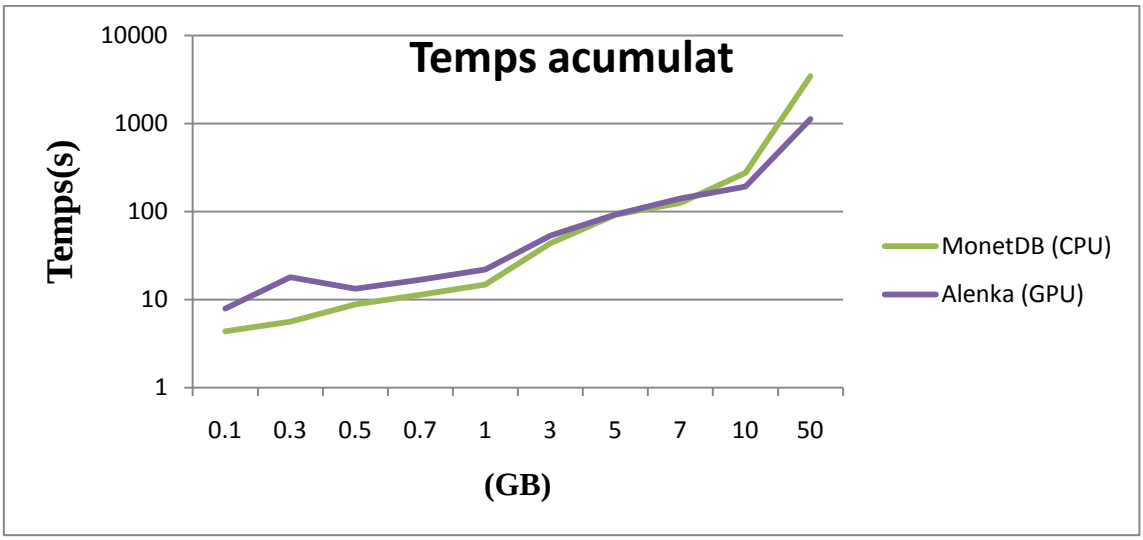


Figura 4.6: Gràfica de la suma de temps de les 6 consultes descartant els dos sistemes orientats a la transacció, per diferents mides de bases de dades. Escala logarítmica.

Tot seguit es mostren les gràfiques en detall de cada consulta (Figures de 4.7 a 4.12). Com es podrà observar, es poden formar dos grups a partir de les tendències de cada consulta. El primer, format per la Q1, Q4 i Q6, segueix exactament el mateix patró que en els gràfics de les mitges. MonetDB ofereix més bon rendiment que Alenka per mides de problema petits però en el moment en que el volum de dades creix, s'acaba degradant molt més i es produeix una diferencia observable.

El segon grup, format per Q2, Q3 i Q5 es diferencia en que finalment, amb volums grans de dades, els dos sistemes ofereixen un rendiment molt pròxim, i s'han de destacar a part ja que no son observables en les gràfiques de les mitges.

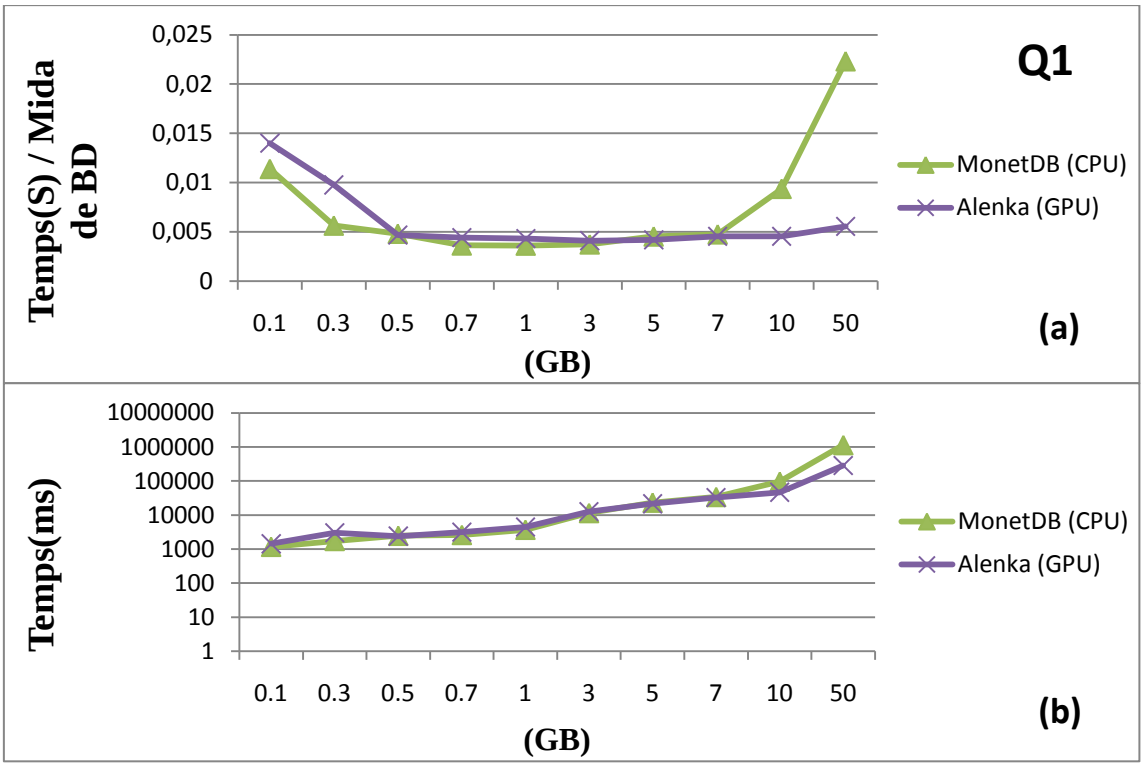


Figura 4.7 : Gràfica de la consulta 1 detallada per MonetDB i Alenka. (a) Escala en Temps(s) / mida BD. (b) Escala logarítmica.

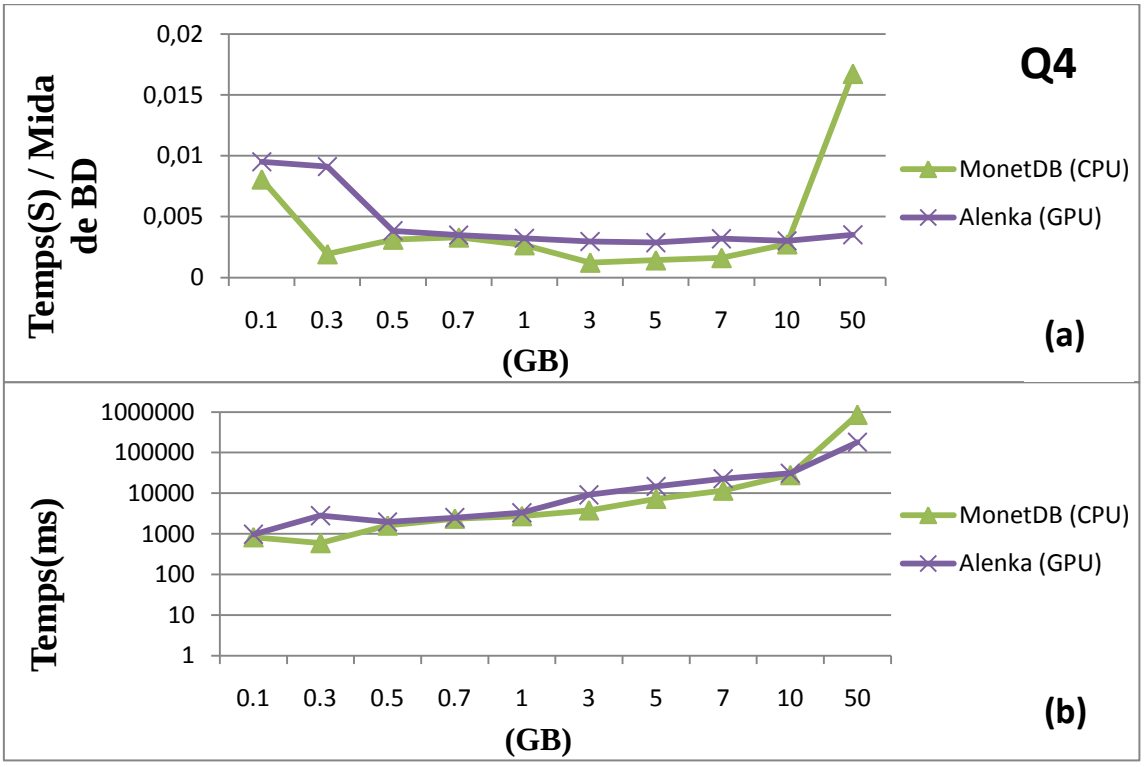


Figura 4.8 : Gràfica de la consulta 4 detallada per MonetDB i Alenka. (a) Escala en Temps(s) / mida BD. (b) Escala logarítmica.

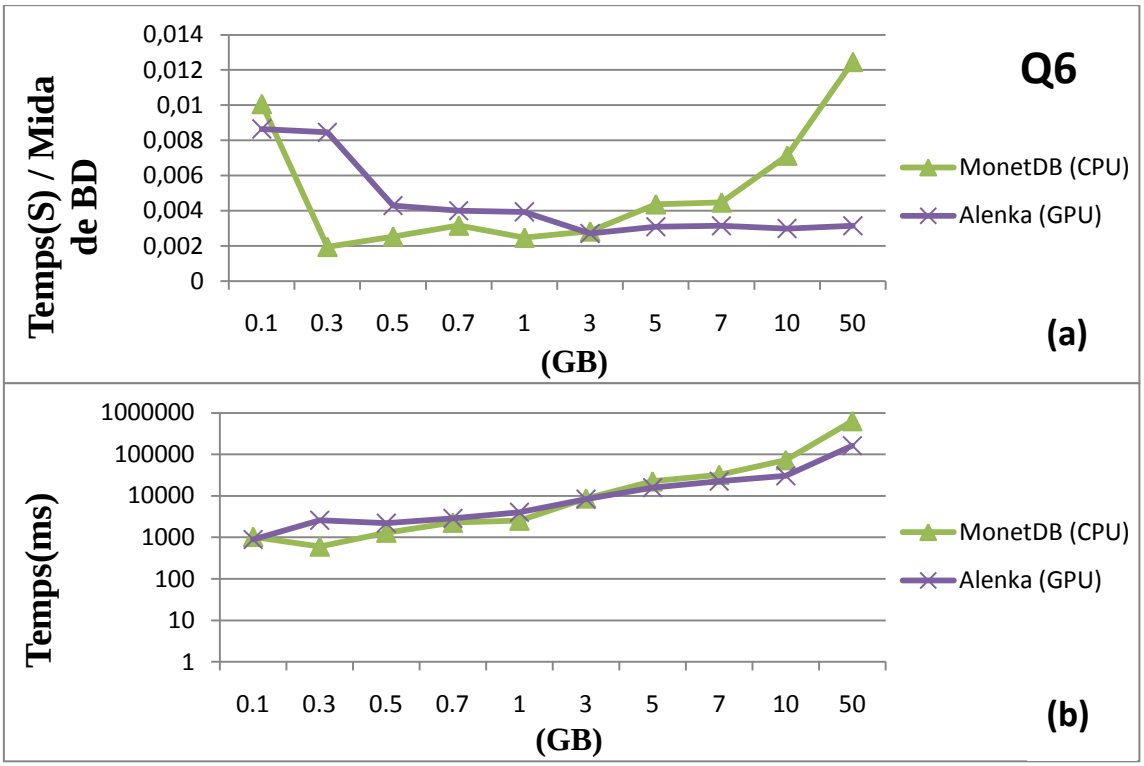


Figura 4.9 : Gràfica de la consulta 6 detallada per MonetDB i Alenka. (a) Escala en Temps(s) / mida BD. (b) Escala logarítmica.

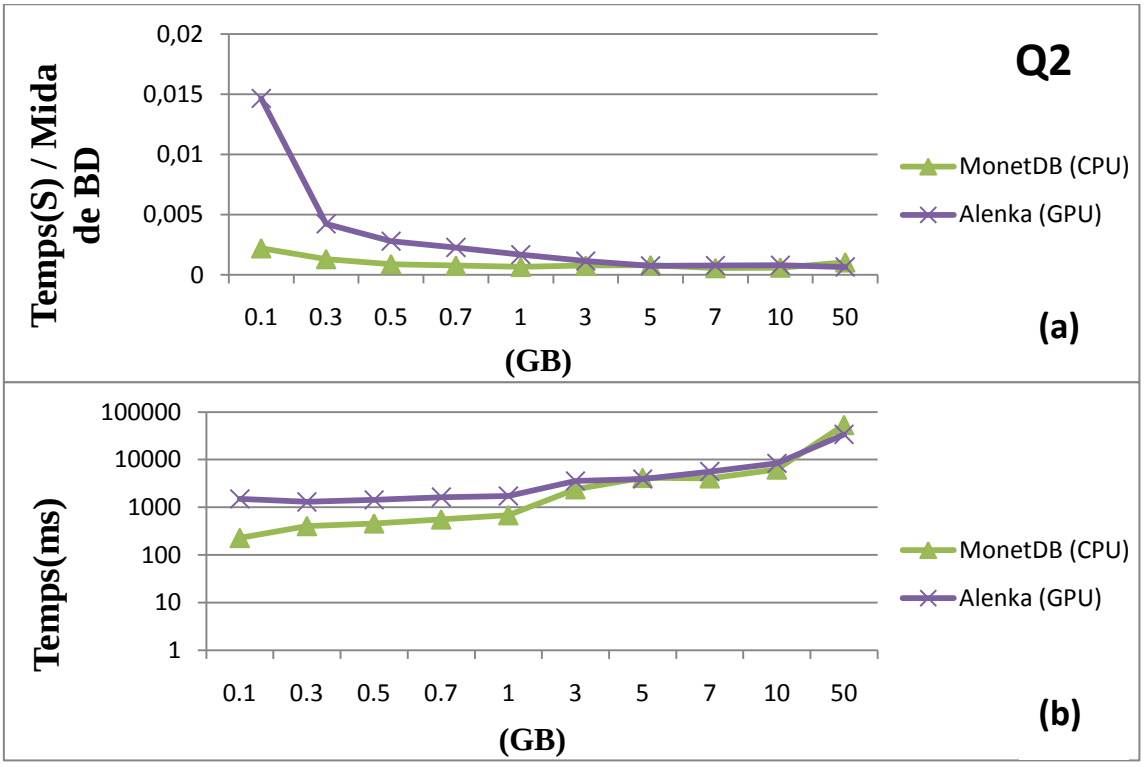


Figura 4.10 : Gràfica de la consulta 2 detallada per MonetDB i Alenka . (a) Escala en Temps(s) / mida BD. (b) Escala logarítmica.

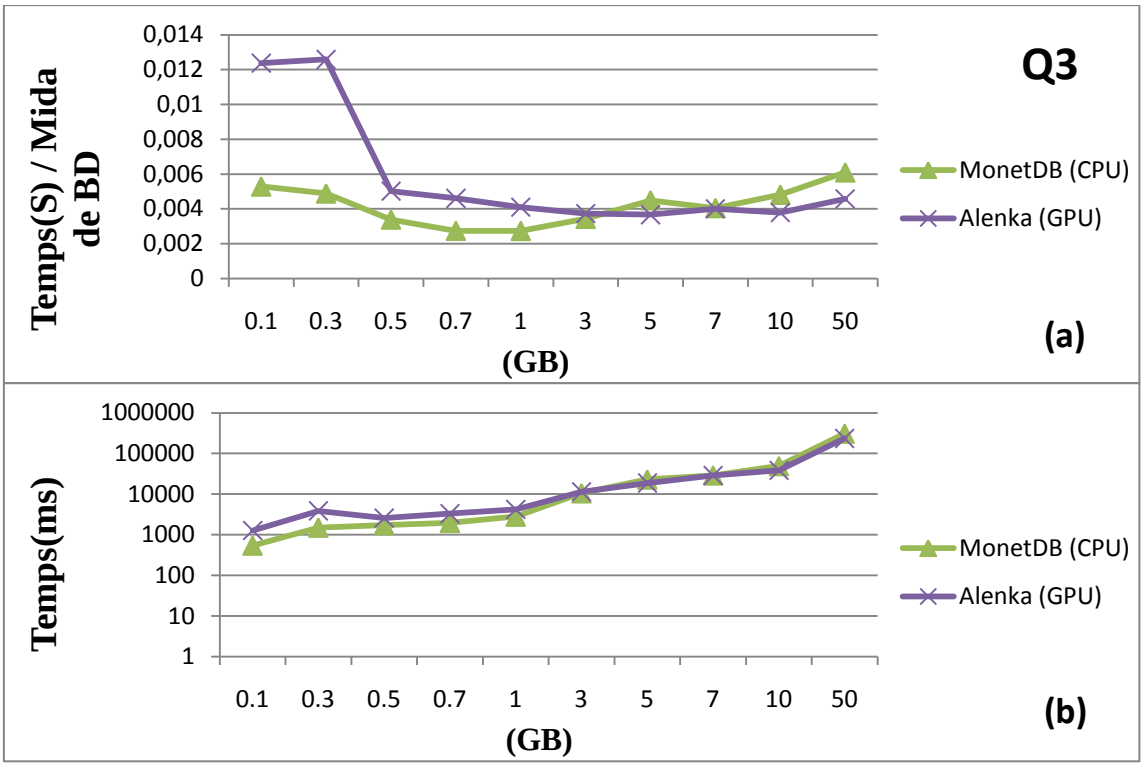


Figura 4.11 : Gràfica de la consulta 3 detallada per MonetDB i Alenka. (a) Escala en Temps(s) / mida BD. (b) Escala logarítmica.

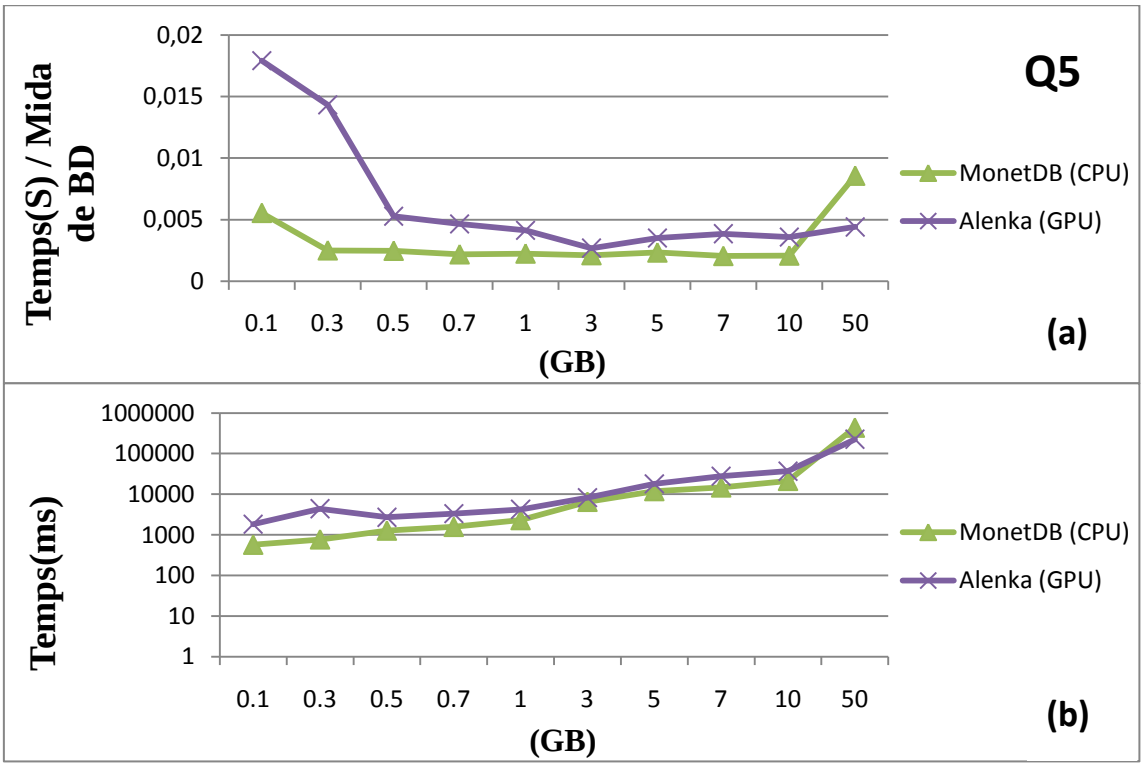


Figura 4.12 : Gràfica de la consulta 5 detallada per MonetDB i Alenka. (a) Escala en Temps(s) / mida BD. (b) Escala logarítmica.

La possible explicació d'aquests dos comportaments diferents es podria obtenir observant els esquemes dels plans d'execució sense optimitzar d'aquestes consultes. A continuació (Figura 4.13 i 4.14) es presenten dos d'ells com a representatius de cada un dels dos grups, la resta es poden consultar en l'Annex 8.2.

Figura 4.13 : Pla d'execució sense optimitzar de la consulta 6. Es mostra en representació dels plans d'execució de les consultes Q1,Q4 i Q6.

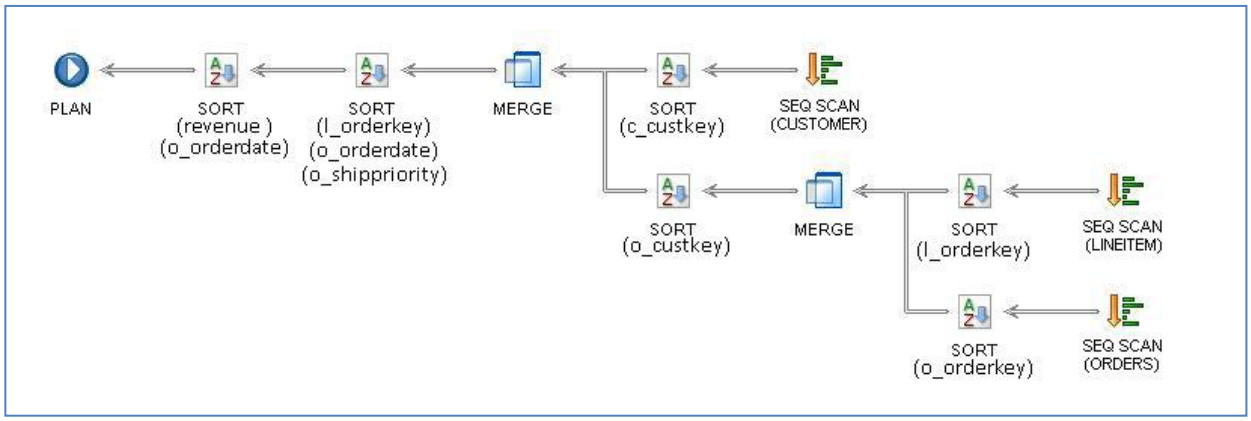


Figura 4.14 : Pla d'execució sense optimitzar de la consulta 3. Es mostra en representació dels plans d'execució de les consultes Q2,Q3 i Q5.

En Q1, Q4 i Q6 les operacions que s'han de realitzar són molt semblants i impliquen només 1 o 2 taules. Sempre es realitza, en primer lloc, un recorregut en seqüència de la taula LINEITEMS (on s'hi guarden aproximadament el 65% de les dades). Aquesta operació no es pot posposar mai i sempre ha de ser la primera operació a realitzar (en Q1 i Q6 és la única). Possiblement, en els casos citats (més de 7GB), aquestes dades ja no caben a la memòria RAM i haurà de realitzar les seves operacions al disc a través de memòria virtual, fet que reduirà el seu rendiment

En canvi per l'altre grup, Q2,Q3 i Q5, també s'hi inclou aquesta operació però en totes elles s'involucren 3 o més taules. A més a més, l'optimitzador tindrà la possibilitat de retardar l'operació el màxim possible per reduir els accessos sobre aquesta taula gran.

Aquesta es una tècnica típica dels optimitzadors, ja que si es comença l'execució de les operacions per les taules més petites, es pot reduir els futurs accessos sobre les taules grans, ja que s'hauran eliminat el nombre de registres a buscar.

Tot això tenint en compte que el pla d'execució real de MonetDB no ha pogut ser estudiat, ja que per comprendre'l en profunditat s'haurien d'entendre les desenes d'operacions implementades en aquest. Es pot dir però, que s'ha pogut comprovar en Alenka que per aquestes mides de problema mai es supera la memòria real disponible

per la mida de bloc triat, i per tant aquest sistema mai haurà d'anar a fer operacions al disc.

També cal fer notar, que Alenka sempre produeix una tendència en gràfica més previsible i lineal, amb menys pics. Això es un tret a destacar ja que aquesta previsió pot ser útil en producció.

5 Estudi dels resultats d'Alenka

En aquest apartat es realitzarà un estudi en profunditat dels resultats precedent a les conclusions. Primerament es comentaran els valors de tots els SGBD utilitzats. A continuació ens centrarem en els dos més ràpids i finalitzarem amb un estudi individual d'Alenka.

5.1 Implementació d'Alenka en primitives paral·leles

A continuació es mostrarà la implementació de les operacions SQL bàsiques realitzades en GPU amb les primitives de la llibreria Thrust. S'utilitzarà la dades de la figura 5.1 com a dades d'exemple.

Linies_Comandes

ID	Order_ID	Article_ID	Quantitat	Preu
1	1	5	10	50,25
2	1	3	13	5,75
4	2	1	25	10,99
8	3	5	10	50,25

Article

ID	NOM	STOCK
1	Pantalons	5
3	Gorra	7
4	Jaqueta	4
5	Camiseta	10

Figura 5.1 : Esquema i dades de les dos taules Linies_Comandes i Article usades en els exemples posteriors.

Order by

Sintaxis: ORDER BY columna1 [ASC|DESC] [, ... columnaN [ASC|DESC]]

Operació que ordena un conjunt resultant a partir d'unes determinades columnes. Si se n'especifica més d'una columna, sempre tindrà més prioritat l'atribut de més a l'esquerra en la comanda. Els valors ASC i DESC es refereixen a ascendent o descendent. Les operacions a realitzar en primitives Thurst són:

Sequence: Generem un vector de mida igual al nombre de files del conjunt, i que s'incrementi de 1 fins a n. Aquest vector s'utilitzarà com a vector de permutacions i l'anomenarem d'aquesta forma.

Per cada columna que es vol ordenar en ordre invers en prioritat, s'itera de la forma següent:

Gather: S'aplica a la columna un gather a partir del vector de permutacions actual. La primera iteració no aplica cap canvia a la columna.

Stable_Sort_By_Key : Es realitza un stable_sort_by_key, on el vector de claus son els valors de la columna i el vector de valors es el vector de permutacions. D'aquesta forma

en aquest últim vector tindrem les noves permutacions a aplicar segons la ordenació feta en la clau.

Un cop finalitzat, al vector de permutacions ja tenim els nous ordres que han de tenir totes les columnes del conjunt i només s'hi ha d'aplicar un gather a totes elles.

Gather: S'aplica el gather per a cada columna de tot el conjunt.

Per exemple si volem ordenar la taula Articles de la forma ORDER BY NOM ASC, PREU ASC, es fa de la forma següent:

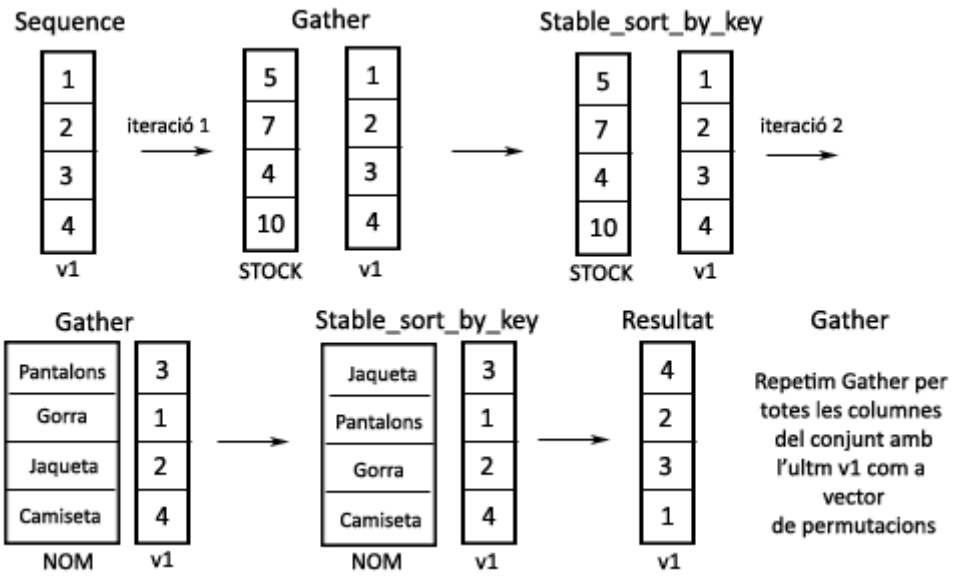


Figura 5.2 : Exemple de l'operació ORDER BY.

Group By

Sintaxis: GROUP BY columna1 [, ... columnaN]

Operació que agrupa les files d'un conjunt que tenen el mateixos valors de les columnes especificades. Si se n'especifica més d'una columna, sempre tindrà més prioritat l'atribut de més a l'esquerra en la comanda. La mateixa operació sol ser aprofitada per fer càlculs sobre les files agrupades, com podria ser el calcular el màxim o la suma de cada agrupació. Les operacions a realitzar en primitives Thurst són:

Realitzem primerament una ordenació amb les mateixes operacions que l'ORDER BY.

Inicialitzem amb **Sequence** un vector de tot zeros. Aquest vector acumularà els punts d'agrupació, i l'anomenarem acumulador.

A continuació apliquem, per totes les columnes que volem agrupar:

Transform: Apliquem transform amb l'operació not equal i com a paràmetres d'entrada Columna[0..N-2] i Columna[1...N-1]. D'aquesta forma obtenim un vector dels punts de les columnes on els valors deixen de ser iguals.

Transform: Amb l'operació logical_or i com a paràmetres d'entrada el vector del transform anterior i el vector acumulador, que també s'utilitzarà per guardar els resultats. D'aquesta forma anem afegint nous punts de tall al vector acumulador.

Amb aquestes iteracions tindrem un vector acumulador que ens indicarà per quins índex s'ha de realitzar l'agrupació.

Per finalitzar:

Inclusive_Scan: Realitzem un inclusive_scan sobre el vector acumulador.

Transform: Amb la operació odd_even que aplica un "mod 2" a tots els valors del vector acumulador. Desem el resultat en un altre vector, ja que aquest s'utilitzarà en futures segmented scans.

Transform: Amb l'operació minus_nz que retorna 1 si (x-y) != 0. En cas contrari retorna 0. Els paràmetres d'entrada es el mateix vector acumulador, de la forma acumulador[0..N-2] i acumulador[1..N-1]. El resultat li posem un 1 al final del vector i el guardem, ja que serà un segon vector que s'utilitzarà per fer agrupacions.

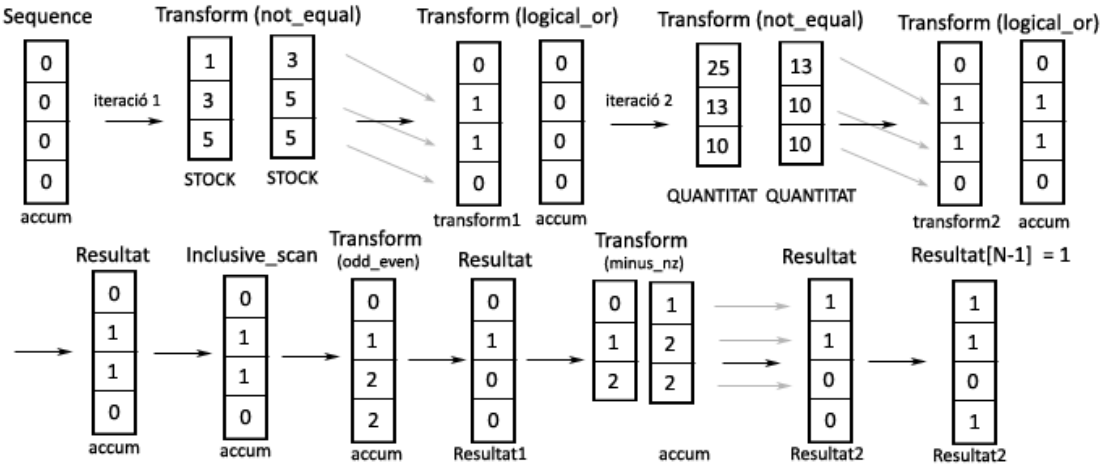


Figura 5.3 : Exemple de l'operació GROUP BY.

Un cop tenim Resultat1 i Resultat2 només els hem d'utilitzar consecutivament per poder realitzar les operacions d'agrupació. Si es vol realitzar un *count* dels valors que es poden agrupar hem de realitzar un Sequence, Inclusive_Scan_by_key i Copy_if, mostrats a continuació en la figura 5.4.

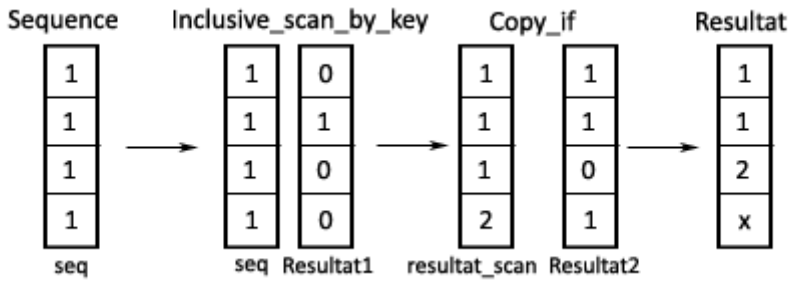


Figura 5.4 : Exemple de l'operació *count* amb els resultats del GROUP BY.

Si es vol realitzar una suma de tots els valors de l'agrupació es pot fer igualment però eliminant el sequence, i es pot veure a la figura següent.

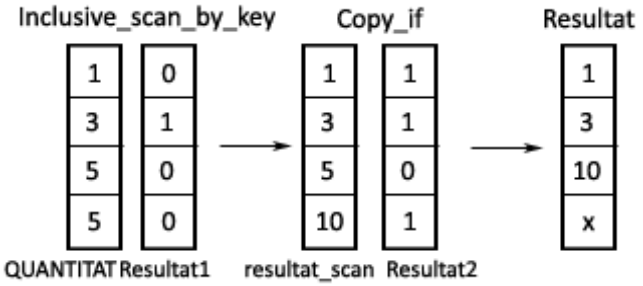
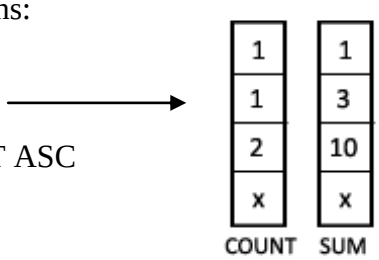


Figura 5.5 : Exemple de l'operació *sum* amb els resultats del GROUP BY.

Així es pot veure com es realitzen les agrupacions:

```
Select count(*), sum(QUANTITAT)
from LINIES_COMANDA
GROUP BY ARTICLE_ID ASC, QUANTITAT ASC
```



Filter

Sintaxis:

```
WHERE
columna1 [ = , < > , >= , <= ] valor
[ ... (AND|OR) columna2 [ = , < > , >= , <= ] valor ]
```

Operació que filtra les files d'una conjunt a partir d'un seguit de criteris de recerca. Aquestes operacions es realitzen ràpidament sobre GPU a través de:

Sequence: Es genera un vector on tots els seus valors son els de recerca.

Transform: S'aplica amb l'operació equal, grather , less, grather_equal, less_equal segons el criteri de recerca. Els dos vectors d'entrada son el vector generat pel *sequence* i la columna per la que es busca.

Si hi ha varis criteris de recerca s'hauran d'iterar amb *Transform* (logical_OR, logical_AND) per unir-los en un únic vector, on hi haurà les posicions de les files que compleixen tots els criteris de recerca.
Si per exemple es busca tots els articles que tenen un *stock* <= 8 es realitza de la següent manera:

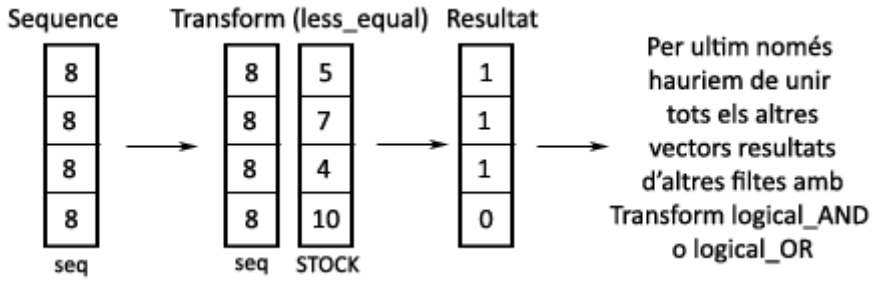


Figura 5.6 : Exemple de l'operació FILTER.

Join

Sintaxis: SELECT * FROM taula1 JOIN taula2 on taula1.colunma1 = taula2.columna1

Operació que uneix les files de dues taules només si es compleix que els valors de la columna de la taula1 son també a la taula2. Les operacions a realitzar en primitives Thurst són:

Es realitza un ORDER BY de la taula més petita per la columna triada.

A continuació, es fa un **lower_bound** i un **upper_bound**, per mirar on podríem insertar els valors del vector no ordenat dins el vector ordenat. D'aquesta forma obtindrem les posicions que coincideixen.

Transform: Amb la operació minus (resta). Els paràmetres d'entrada son els dos vectors anteriors.

Exclusive_scan: Sumem els valors del vector *transform* anterior.

For_each: Aquesta es la única primitiva no implementada pensant en la paral·lelització. Això es degut a que aquesta primitiva aplica una funció amb un iterador. No mostrada aquí, és un bucle per trobar les posicions de les dos columnes que coincideixen.

Per exemple, si volem fer una JOIN de les taules LINIES_COMANDES i ARTICLES per la columna article_id farem:

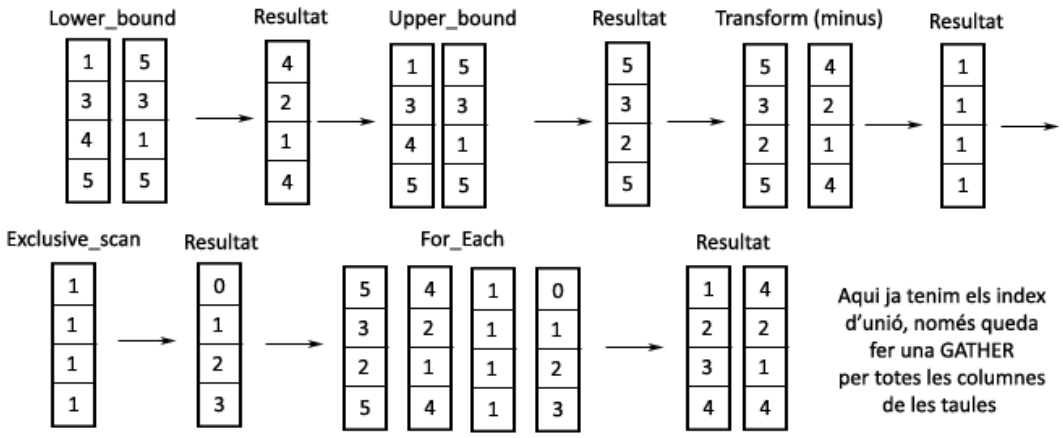


Figura 5.7 : Exemple de l'operació JOIN.

Per concloure, es vol fer notar la gran adaptació de les operacions SQL en primitives paral·leles que justifiquen els bons resultats obtinguts.

5.2 Estudi dels resultats d'Alenka

En aquest apartat estudiem els resultats individuals d'Alenka. Es fa un estudi de perfil per veure si es poden obtenir millores en el futur. Tots els futurs estudis es realitzen sobre la mida del problema de 50 GB.

Cal destacar els bons resultats obtinguts en comparació amb els altres sistemes quan la mida del problema es gran.

Així doncs, un cop vistos els resultats finals, es volen desglossar per cada tipus d'operació interna. Ho aconseguirem a través de les eines de profiling i es mostren a continuació:

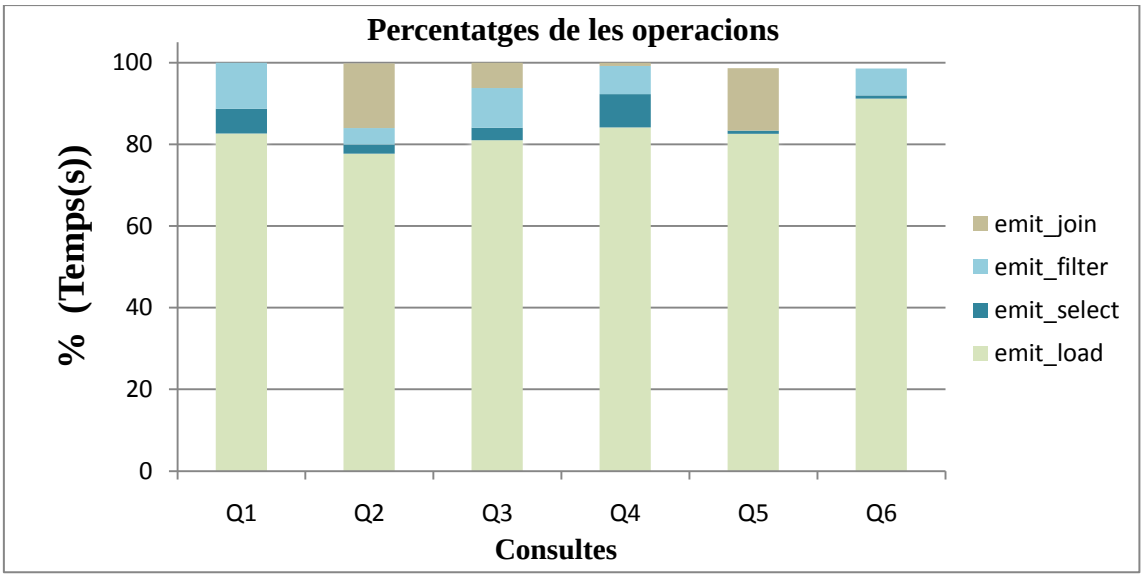


Figura 5.8 : Taula on s'indica el percentatge de temps sobre el total de les funciona més importants del codi. Cas on la mida del BD es de 50GB.

En aquesta gràfica (Figura 5.9) es pot veure com el percentatge de temps utilitzat en lectures a disc (emit_load) és molt gran. Això fa intuir que en aquest moment la implementació està limitada per E/S a disc. Es pot veure també que les operacions paral·leles fetes a GPU ocupen aproximadament el 20% del temps total, amb JOIN i FILTER com a operacions més costoses.

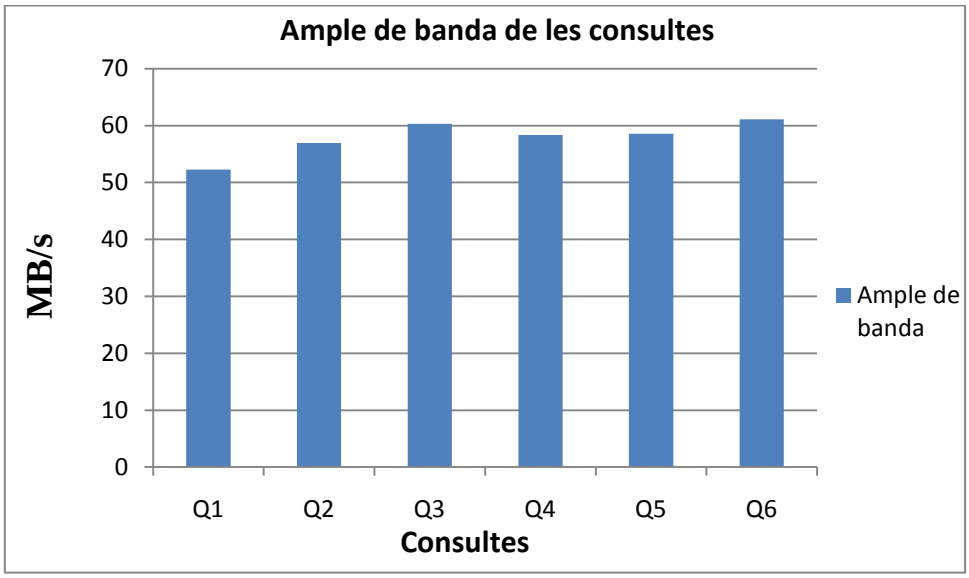


Figura 5.9 : Gràfica dels amples de banda reals obtinguts per cada consulta. Cas on la mida del BD es de 50GB.

També cal destacar la instrucció *CudaMallocHost*. Aquesta, acaba sent bastant costosa i és una de les responsables, en alguns casos, de que els temps del profiling siguin superiors a les del següent estudi del punt 5.3. Això és degut a que s'aconsegueix reservar espai a memòria de CPU que serà accessible directament per la GPU. D'aquesta forma, posteriorment s'aconseguirà reduir el temps de transferència de dades entre CPU i GPU. La instrucció *malloc* també pot ser utilitzada però després els temps de transferència es major. Tot i així caldria aprofundir sobre aquesta instrucció en el futur.

5.3 Resultats de les proves de disc

Intuint que el codi d'Alenka està limitat pel disc, com s'ha pogut veure en els percentatges obtinguts amb els programes de profiling, es va voler confirmar utilitzant la base de dades de mida major (50 GB). Així doncs, primerament es va realitzar un prova de disc per obtenir la taxa de lectura en MB/s en el pitjor i millor dels casos.

Per tant es va crear un fitxer de 50 GB que acumularà fragmentació, i es van realitzar tres lectures del fitxer complert obtenint un temps. Un cop feta la mitja, s'obté una taxa de lectura de 61,58 MB/s. D'altre banda també s'ha obtingut una taxa de lectura seqüencial sense fragmentació de 109,5 MB/s a través d'un fitxer de 4GB. Aquests valors s'han confirmat estar pròxims a la taxa mínima i màxima del disc usat segons [11].

Un cop tenim aquests valors, s'ha calculat el volum de dades que cada consulta utilitza en la base de dades d'escala 50GB, obtenint el percentatge aproximat del temps que l'executable estarà llegint a disc i no processant.

	Mida de dades (GB)	**% real (profiling)**	**% estimat (61,58 MB/s)**	**% estimat (109,5 MB/s)**
Q1	11,71	82,69	68,31	38,42
Q2	1,60	77,69	78,80	44,32
Q3	11,28	81,05	79,47	44,70
Q4	8,85	84,19	81,57	45,88
Q5	10,71	82,57	78,48	44,14
Q6	8,92	91,18	91,93	51,70
Mitja	8,85	83,23	79,76	44,86

Taula 5.10 : Taula dels percentatges de temps estimat al disc mínim i màxim de cada consulta sobre el temps total d'execució, juntament amb la mida de dades tractada en cada consulta i el percentatge real vist en la Figura 5.9 en forma d'ample de banda.

Com es pot veure els percentatges de temps a disc concorden bastant amb els obtinguts amb el profiler. Així que podem dir que el percentatge real serà proper al % màxim, inclòs el supera segons en alguns casos segons el profiler. Això es degut a que totes les columnes estan guardades en diferents fitxers i per tant no es podran realitzar grans lectures seqüencials

Per tant, amb totes aquestes dades podem afirmar que el rendiment del codi esta limitat per disc. Com a conseqüència, com es veurà a les conclusions, s'hauran de realitzar optimitzacions del codi per poder reduir aquests percentatges i aconseguir un millor rendiment.

6 Conclusions i possibles ampliacions

A continuació es fa una valoració general del projecte realitzat, tractant els resultats obtinguts pel problema de les consultes analítiques i la viabilitat de la tecnologia de GPU sobre aquests.

En primer lloc, cal indicar que s'han complert els objectius que es van plantejar al inici del projecte pels motius que es descriuen a continuació.

L'objectiu principal que es va marcar al començament del projecte va ser entendre i utilitzar la tecnologia GPU, amb els seus avantatges i inconvenients. Un cop finalitzat el projecte puc dir que s'ha vist l'arquitectura actual de les GPUs, la seva forma de treballar i com poder treure un bon rendiment d'aquesta tecnologia.

Per poder-ho fer, s'ha d'aconseguir transformar l'estructura de dades del problema a un format que s'adapti bé als patrons paral·lels. En el codi vist en aquest projecte s'agafa el format vectorial, que és pel que es van orientar les GPUs en els seus inicis i amb el que millor rendiment s'aconsegueix. Utilitzant vectors s'han trobat implementades llibreries que ofereixen primitives paral·leles optimitzades (correctes accessos a memòria, minimització de les instruccions atòmiques i de sincronisme, etc). D'aquesta forma si s'aconsegueix reescriure el codi CPU mitjançant aquestes primitives s'aconseguirà obtenir un bon paral·lelisme i, en principi, passarem a dependre de la mida del problema per poder obtenir bons guanys en temps.

La quantitat d'informació que s'enviarà de la CPU a GPU serà determinant per saber si és favorable realitzar el processament sobre GPU. En aquest cas, s'ha vist que amb unes mides de problema petits no s'aconsegueix superar als altres sistemes CPU. En el moment en que creix el volum de dades, Alenka (GPU) esdevé el millor pel que fa a temps d'execució.

Un segon objectiu que em vaig marcar, va ser poder entendre amb més profunditat els sistemes de bases de dades orientats a la transacció (Firebird i PostgreSQL). En particular, m'interessava veure les funcionalitats que ofereixen a l'hora de dissenyar bases de dades, com poden ser les regles d'integritat o els diferents tipus d'índexs. També volia entendre i detectar quan es selecciona un tipus d'algorisme o altre depenent del pla d'execució.

Al finalitzar el projecte s'ha pogut configurar aquests dos sistemes mencionats perquè s'adaptin correctament als recursos hardware disponibles. Per tant, s'ha hagut d'entendre els seus paràmetres de configuració i quins efectes provocaven en la seva modificació. També s'ha aconseguit llegir i entendre els seus plans d'execució per poder identificar els motius pels quals una consulta no es comporta com hauria de fer-ho.

Per poder unir els interessos personals en bases de dades juntament amb les GPU, es va haver de buscar un problema específic que em permetés involucrar els dos temes. Així doncs, després d'escollir un projecte de recerca de bases de dades amb GPU es va veure que el problema a tractar seria el de les consultes analítiques. Alenka ja incorporava un seguit de consultes del *benchmark* TPC-H, orientat a realitzar consultes complexes en el menor temps possible. D'aquesta forma només es va haver de seleccionar un tercer sistema de bases de dades en CPU que oferís bons resultats amb aquest tipus de joc de proves per poder realitzar la comparació amb el sistema GPU.

Finalment es pot dir que durant el projecte s'ha pogut avaluar quatre sistemes diferents a través d'un *benchmark* estàndard, juntament amb les seves eines de creació i execució. D'aquesta forma s'ha vist com treballar fixant un entorn de treball inicial, fer una avaluació i finalment realitzar un anàlisi de resultats per poder extreure'n les conclusions pertinents. Per realitzar aquest últim punt, s'ha hagut de treballar amb eines de *profiling* que m'han ajudat a trobar els punts dèbils del codi avaluat.

Per últim, voldria concloure que un cop s'ha vist els resultats obtinguts amb la tecnologia GPU en bases de dades i la informació obtinguda durant la recerca, puc dir que podria ser utilitzada en entorns reals de producció per aquests tipus de problemes. A més a més, la tendència vista és que els sistemes de bases de dades que aborden les consultes analítiques tenen com a propostes de futur incloure les GPUs en les seves implementacions.

6.1 Línies obertes

Les línies obertes que apareixen sobre la implementació de GPU son varies. En primer lloc es considera interessant intentar reduir el percentatge de temps de les lectures de disc. Per això, en el punt 6.1.1 es presenta un breu estudi d'un esquema de compressió utilitzat per MonetDB i que podria ser útil en Alenka.

També s'ha vist que cal millorar la gestió de memòria GPU del codi d'Alenka. Això es degut a que els conjunts de dades es transfereixen a GPU per fer operacions i els resultats es retornen a memòria CPU, eliminant els conjunts de GPU. Tot i així, seria convenient trobar la forma de mantenir, si fos possible, els conjunts en la GPU si s'utilitzen en alguna operació posterior. D'aquesta forma es reduirien les transferències entre CPU i GPU, i que solen ser costoses.

6.1.1 Compressió de dades

Les primeres avaluacions fetes sobre el codi font d'Alenka indiquen que el temps dedicat a la lectura de dades a disc es un dels punts a millorar, ja que es un tant per cent elevat del total de temps d'execució. Al veure les alternatives utilitzades per altres SGBD de codi lliure, per accelerar l'execució i intentar solucionar el problema del coll de botella de les lectures a disc en sistemes d'altes prestacions, s'ha vist que un punt important del que indiquen es la compressió de dades. Això es degut a que es

preferible dedicar temps d'execució del processador a descomprimir dades en comptes de mantenir un mida de dades a disc gran.

Així doncs, al aconseguir els algorismes de compressió aptes per aquests requeriments, s'ha agafat tres algorismes presentats PFOR, PFOR-Delta i PDict (aquest últim de codi privat i no avaluat), que son els utilitzats en MonetDB.

PFOR comprimeix dades enteres, preferiblement valors petits. Actua bé quan els vectors estan parcialment ordenats o agrupats. Per l'algorisme PFOR-Delta es necessari ordenar el vector prèviament, ja que actua sobre les diferencies entre valors consecutius. PDict s'usa a través d'una taula de hash i es útil tant per enters, com per doubles i strings, tot i que es l'únic que no s'ha publicat ja que pertany a codi privat d'una empresa amb SGBD propi.

Per fer les probes inicials s'han realitzat amb quatre vectors de 200 milions de posicions d'enters:

- Per l'algorisme PFOR s'han creat el primer vector (PFOR-1) que conté valors aleatoris entre 1 i 500 sense ordenar ni agrupar i el segon (PFOR-2) que conté valors aleatoris entre 19000000 i 21000000 (interval aproximat en que Alenka representa les dates en format aaaa\mm\dd).

- Per PFOR-Delta s'han creat de forma creixent amb un salt entre cada enter de entre 0 i 20 posicions. El primer (PFOR_DELTA1) partint de 1 i el segon (PFOR_DELTA2) partint de 20000000.

Degut a que els resultats inicials presentats son considerats satisfactoris, s'integrarà amb Alenka per intentar aconseguir guanys amb els accessos a disc. Tot i això, només es considerarà la descompressió en CPU, encara que seria bo aconseguir la descompressió en GPU per alleugerir la transferència de host a targeta.

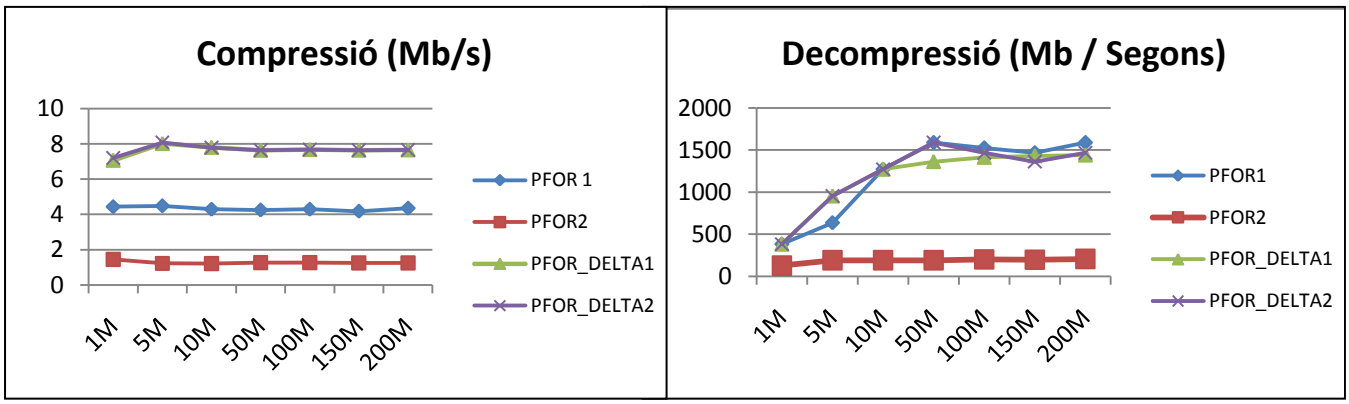


Figura 6.1 : Capacitat de compressió i descompressió de vectors d'integers amb PFOR i PFOR-Delta. (Valors en Mb/s per cada mida de vector)

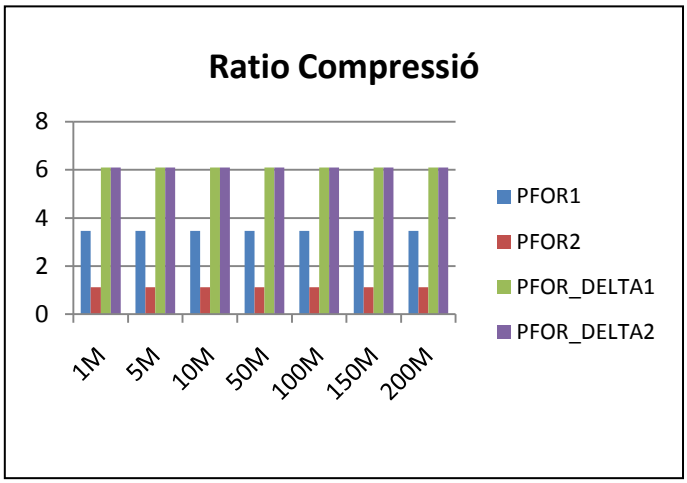


Figura 6.2 : Ràtio de compressió de vectors d'integers amb PFOR i PFOR-Delta. Mateixos valors d'entrada que en figura 6.1.

6.2 Valoració personal

Per finalitzar, es pot dir que realitzar un projecte de llarga durada es molt interessant, ja que en la carrera no es pot arribar a aprofundir a aquest nivell en les practiques realitzades. Per tant es considera que l'actual format del projecte final de carrera ofereix grans avantatges. S'obtenen coneixements que no podria adquirir per altres vies i que, mitjançant el suport del director de projecte, s'aconsegueix plasmar en un treball formal que s'aproxima a una estructura científica de recerca.

Aconseguir involucrar, en aquest treball, objectius i interessos propis que posteriorment hauran de ser avaluats obliga a realitzar un esforç individual per entendre la matèria escollida que sense el projecte final de carrera difícilment es realitzaria. D'aquesta forma al finalitzar s'hauran aconseguit uns coneixements que podran ser utilitzats en el futur.

També ha estat molt interessant poder utilitzar eines noves de profiling d'aplicacions, ja que m'ajudaran a realitzar millor software en el futur. Aquests tipus d'eines que ajuden a trobar els punts dèbils d'un projecte amb molt de codi no s'han vist durant la carrera i faciliten bastant la feina d'anàlisis.

Per finalitzar doncs, puc dir que l'esforç realitzat en aquest treball ha valgut la pena ja que en pocs mesos he pogut veure una gran quantitat de matèria útil per seguir formant-me.

7 Bibliografia

[1] Sandor Heman. Super-Scalar Database Compression between RAM and CPU Cache Master's Thesis Computer Science Centrum voor Wiskunde. 2005

[2] Gregory Smith. PostgreSQL 9.0 High Performance, 2010 Packt Publishing

[3] Helen Borrie. The Firebird Book: A Reference for Database Developers. A-Press (September 8, 2005)

[4] Hector Garcia Molina, Jeffrey D. Ullman, Jennifer Widom. Database Systems, The Complete Book. Second Edition Pearson Internation Edition 2002

[5] http://www.monetdb.org/Home/Features, 5 de Setembre.

[6] Martin Kersten. The MonetDB Architecture. *CWI Amsterdam. 2008*

[7] David B. Kirk , Wen-mei W. Hwu . Programming Massively Parallel Processors: A Hands-on Approach (Applications of GPU Computing Series). Morgan Kaufmann; 1 edition (February 5, 2010)

[8] Jason Sanders**,** Edward Kandrot**.** CUDA by Example: An Introduction to General-Purpose GPU Programming**.** Addison-Wesley Professional; 1 edition (July 29, 2010)

[9] http://sourceforge.net/projects/alenka/, 17 de juny de 2011.

[10] TPC BENCHMARKTM H, Standard Specification Revision 2.14.2. Transaction Processing Performance Council 2011

[11] http://www.tomshardware.com/reviews/hitachi-western-digital-terabyte,2017-2.html, 5 de Setembre.

[12] MonetDB/X100: Hyper-Pipelining Query Execution. Peter Boncz, Marcin Zukowski, Niels Nes. CWI Kruislaan 413 Amsterdam, The Netherlands 2009

[13] Specifying Query Access Plans in the InterBase Optimizer. 2000-2011, IBPhoenix.

[14] Super-Scalar RAM-CPU Cache Compression. Marcin Zukowski, Sandor Heman, Niels Nes, Peter Boncz Centrum voor Wiskunde en Informatica. 2006

8 Annex

8.1 Resultats de comparació de SGBD per a cada consulta

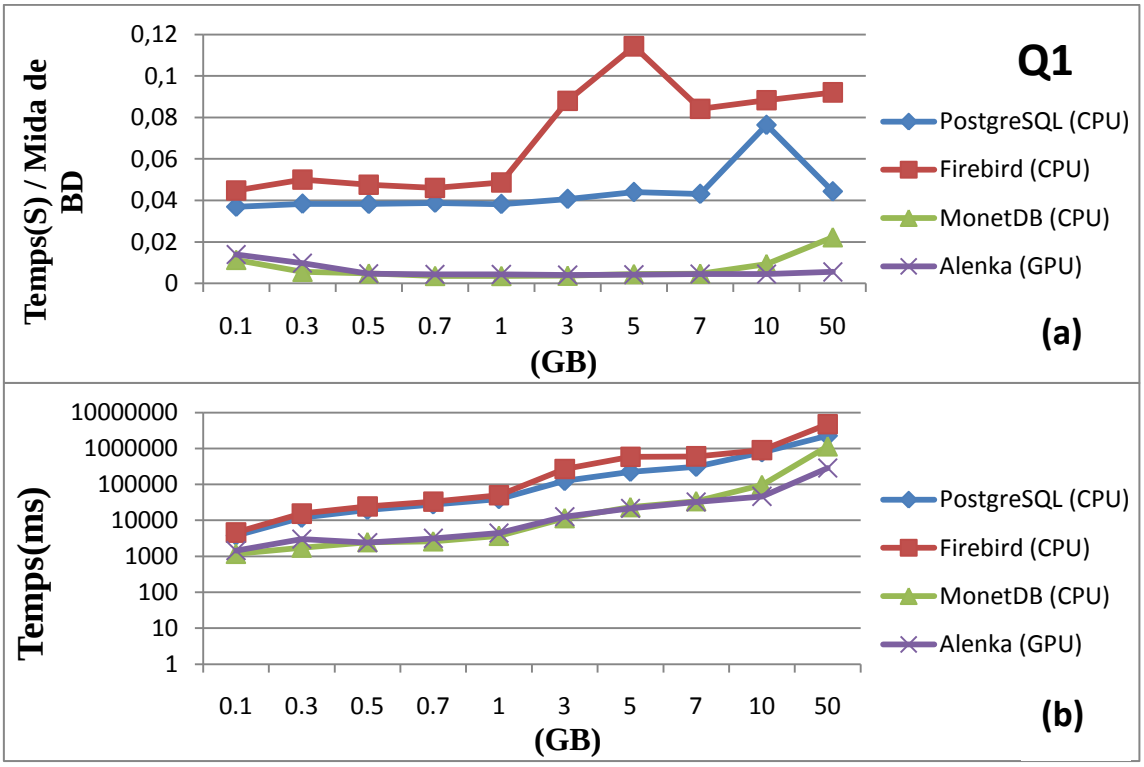


Figura 8.1 : Gràfica de la consulta 1 detallada pels quatre SGBD. Escala logarítmica.

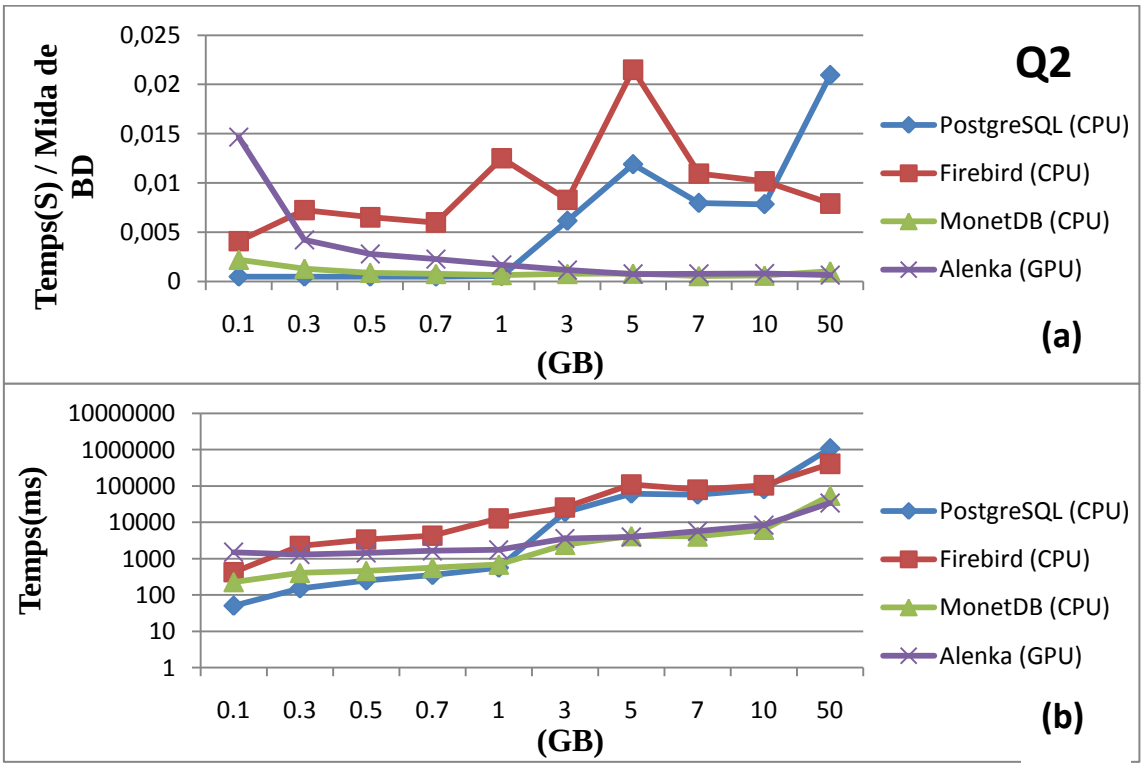


Figura 8.2 : Gràfica de la consulta 2 detallada pels quatre SGBD. Escala logarítmica.

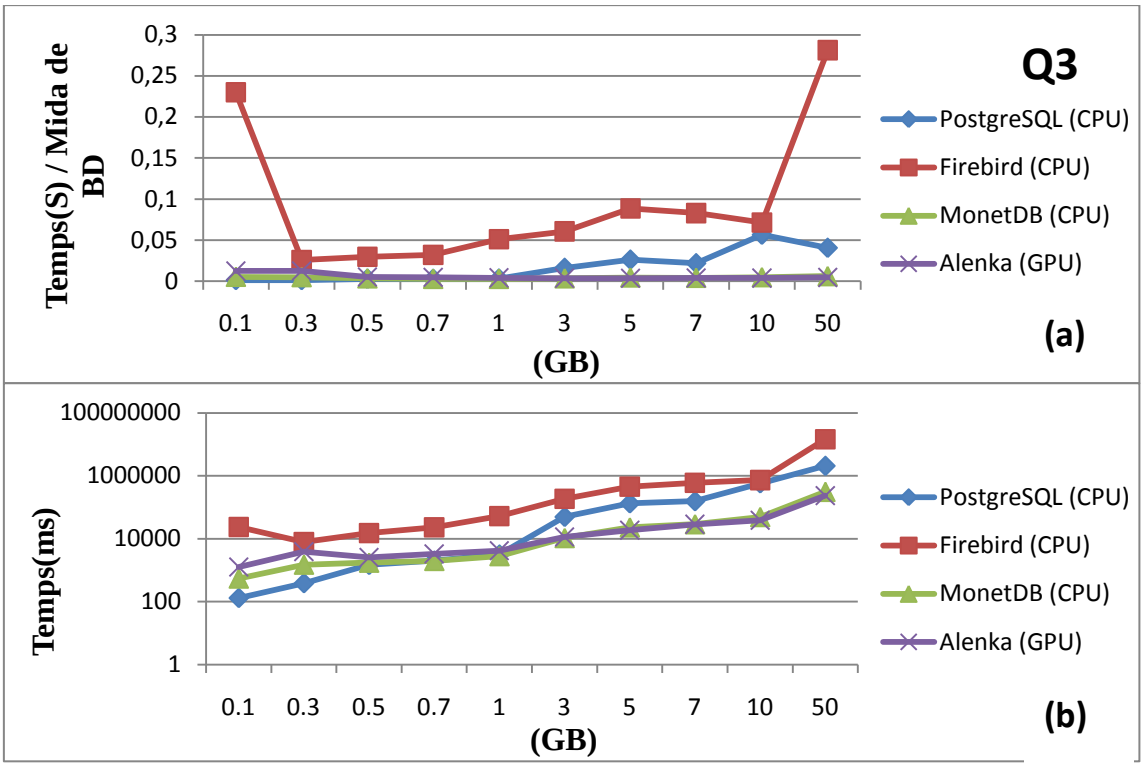


Figura 8.3 : Gràfica de la consulta 3 detallada pels quatre SGBD. Escala logarítmica.

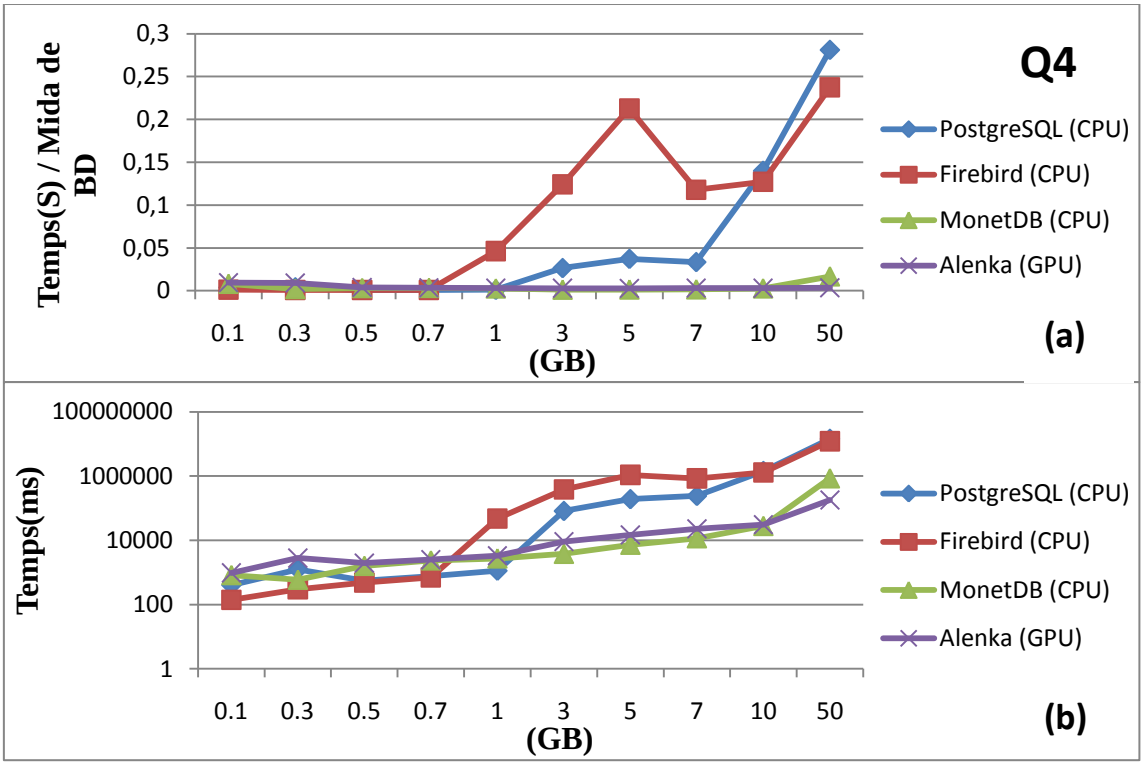


Figura 8.4 : Gràfica de la consulta 4 detallada pels quatre SGBD. Escala logarítmica.

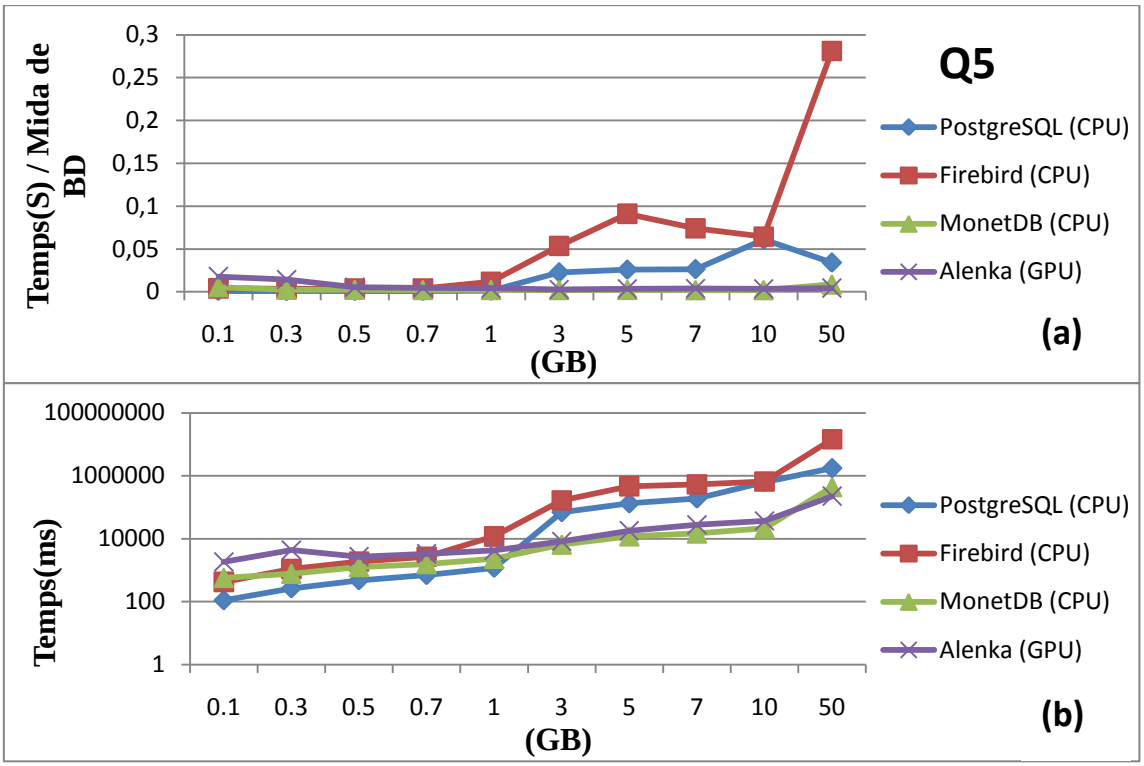


Figura 8.5 : Gràfica de la consulta 5 detallada pels quatre SGBD. Escala logarítmica.

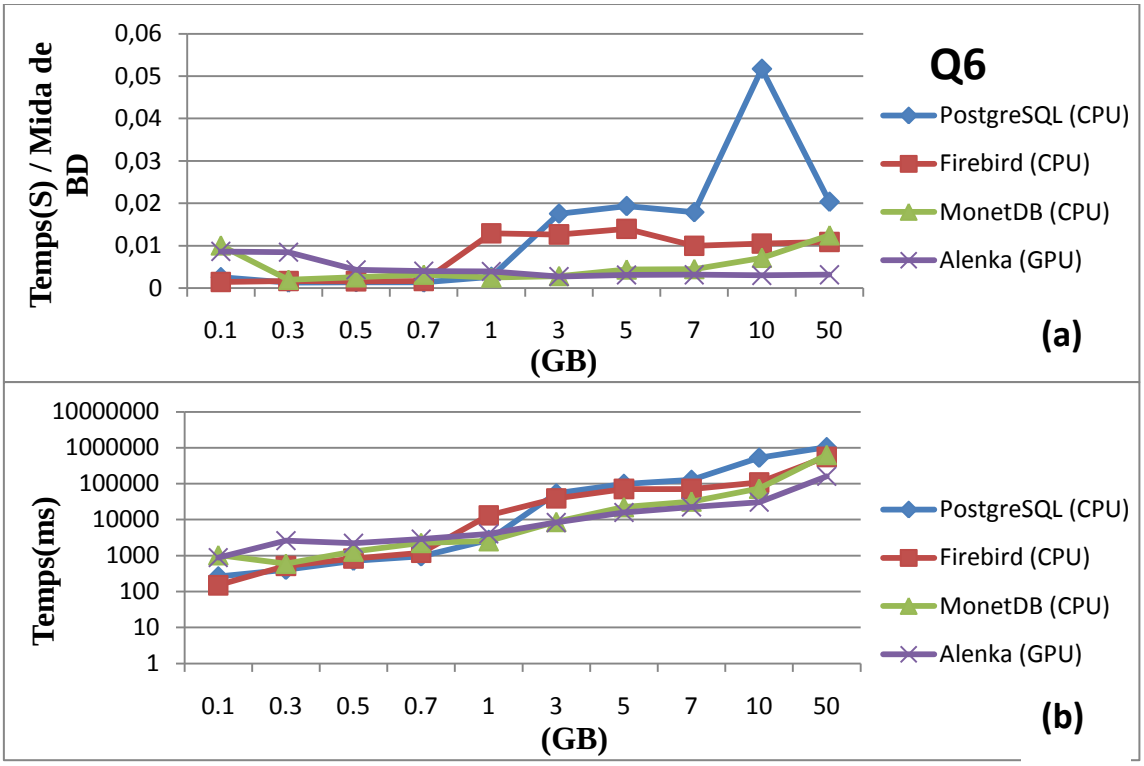


Figura 8.6 : Gràfica de la consulta 6 detallada pels quatre SGBD. Escala logarítmica.

8.2 Plans d'execució de les consultes sense optimitzar

Q1

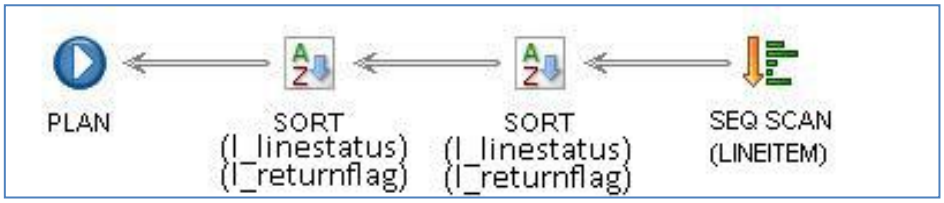


Figura 8.7 : Pla d'execució sense optimitzar de la consulta 1.

Q3

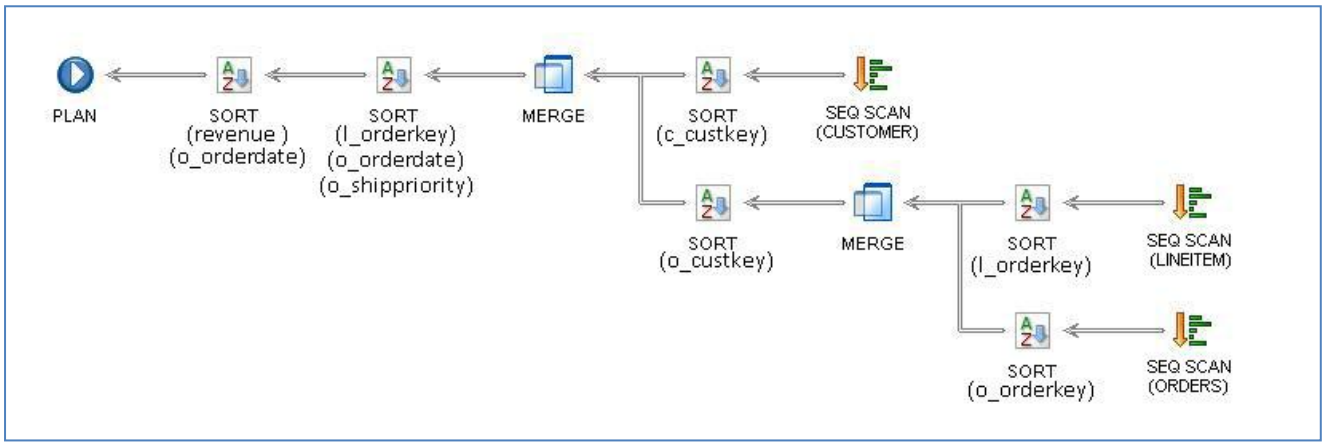


Figura 8.8 : Pla d'execució sense optimitzar de la consulta 3.

Q4

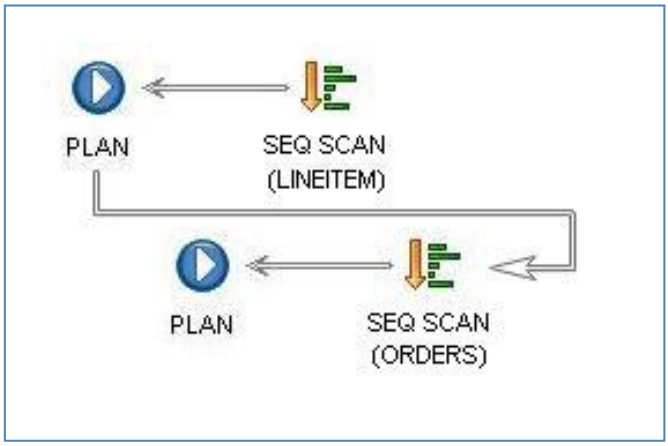


Figura 8.9 : Pla d'execució sense optimitzar de la consulta 4.

Q5

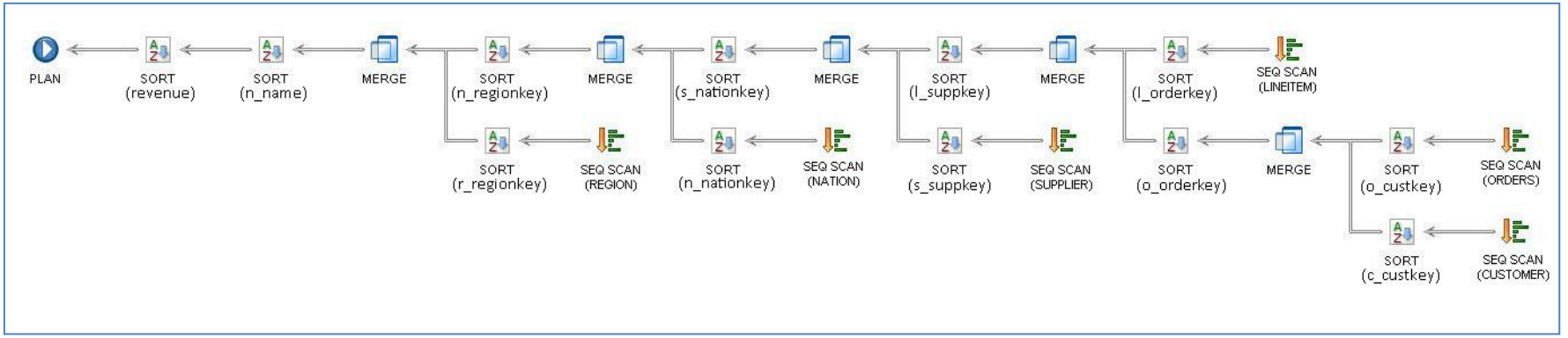


Figura 8.11 : Pla d'execució sense optimitzar de la consulta 5.

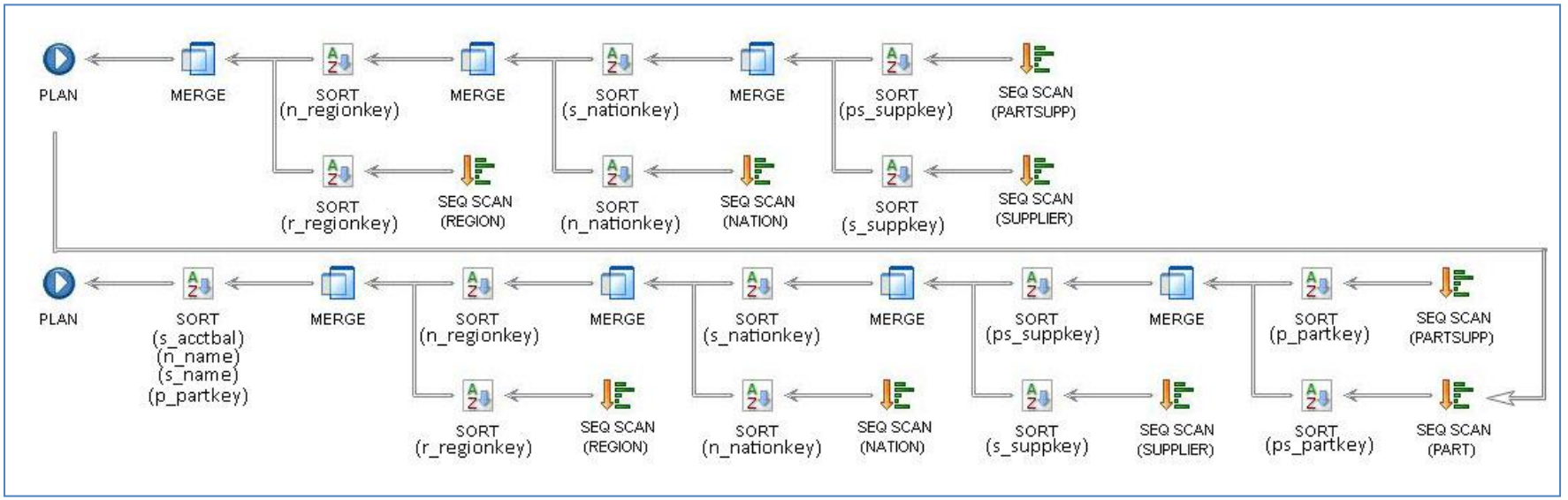


Figura 8.10 : Pla d'execució sense optimitzar de la consulta 2.

Q2

Q6

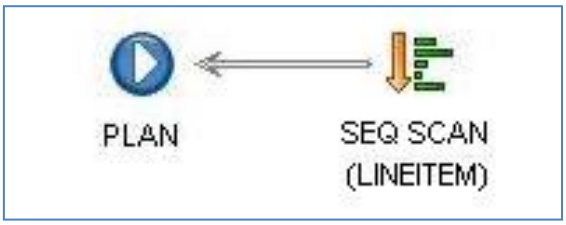


Figura 8.12 : Pla d'execució sense optimitzar de la consulta 6.

Resum

L'objectiu principal d'aquest projecte és avaluar la tecnologia GPU per determinar si pot ser útil en el sector de les bases de dades. En concret s'utilitza el problema específic de les consultes analítiques amb la finalitat de intentar obtenir un temps de resposta més ràpid. Per aconseguir-ho s'executa el benchmark estàndard TCP-H per poder realitzar la comparació entre tres sistemes de gestió de bases de dades CPU amb un altre implementat per GPU.

Resumen

El objetivo principal de este proyecto es evaluar la tecnología GPU para determinar si puede ser útil en el sector de las bases de datos. En concreto se utiliza el problema específico de las consultas analíticas con la finalidad de intentar obtener un tiempo de respuesta más rápido. Para conseguirlo se ejecuta el benchmark estándar TPC-H para poder realizar la comparación entre tres sistemas de gestión de bases de datos CPU con otro implementado para GPU.

Abstract

The main goal of this project is to evaluate GPU technology with the purpose of determining if it can be useful in the database sector. I have chosen the specific problem of the analytical queries to try to reduce their response time. In practice, I have used the standard benchmark TPC-H to compare three database management systems based on CPU with another one implemented for GPU.